# Designing for Forces:
## An Early-Stage Design Program for Axial-Force Structures

by

Alexander D.W. Jordan



Submitted to the
Department of Architecture
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Art and Design

at the

Massachusetts Institute of Technology

June 2011



Signature of Author.....
Department of Architecture
Department of Civil and Environmental Engineering
May 18, 2011

Certified by.....
John A. Ochsendorf
Associate Professor of Building Technology
Thesis Supervisor

Accepted by.....
J. Meejin Yoon
Associate Professor of Architectural Design
Director of the Undergraduate Architecture Program

# Designing for Forces:
## An Early-Stage Design Program for Axial-Force Structures

Alexander D.W. Jordan
Department of Architecture
Department of Civil and Environmental Engineering

**Thesis Advisor**

John A. Ochsendorf
Associate Professor of Building Technology
Department of Architecture
Department of Civil and Environmental Engineering

# Designing for Forces:
## An Early-Stage Design Program for Axial-Force Structures

by

Alexander D.W. Jordan

Submitted to the Department of Architecture on May 18, 2011, in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Art and Design at the Massachusetts Institute of Technology

**Abstract**

Structures that carry most of their load through the axial forces of tension or compression are more materially efficient than standard structures. However, they are not as straightforward to design since the forces in the structure depend on shape. The traditional method of form finding for such axial force structures is to create physical hanging models. These models are slow to produce and difficult to measure. Few digital design aids exist for designing axial force structures, and those that do tend to be for optimization or analysis, not necessarily for early stage design. In addition, they tend to lack desired functionality for a design program, and also tend focus on creating forms without considering engineering functionality. Since form and forces are so intertwined in axial-force structures, consideration of both in the early stages of design is desirable and is not fully addressed by existing programs.

This thesis presents a new early stage design program, ForceDesigner, which improves the functionality of earlier programs and facilitates design by both architects and engineers. It builds on earlier design programs that use the particle spring system for creating digital hanging models, implementing the system in Processing and Java. The result is a program with a number of novel functions that allows designers interested in both form and forces to more quickly and easily create an unlimited number of efficient structures.

**Thesis Supervisor:** John A. Ochsendorf
**Title:** Associate Professor of Building Technology

# Table of Contents

## Acknowledgements

I would like to thank a number of people, without whom this thesis would not be what it is. First, to my advisor, John Ochsendorf, thank you for your guidance and support over the last three years. Most significantly, you gave me the opportunity to achieve a degree in both Architecture and Civil Engineering. To Duks Koschitz, thank you for introducing me to this problem two summers ago, and for being a great mentor. Many of my most lasting lessons were learned not in class, but working with Duks on creating hanging models. I would like to thank everyone who has helped me write ForceDesigner: specifically Rory for help with code design and Taylor for help with swing syntax. Finally, I would like to thank my friends who have made the college experience about more than academics, and my family for supporting me even when I didn't call.

This thesis is dedicated to my father, Barry G. Jordan, MD

## List of Figures

# I. Introduction

## 1.1 Background

The responsibility of architects and engineers entails not only creating functional and beautiful designs, but sustainable ones as well. A major portion of the world's resources are used in construction, thus reducing material use is an important goal for sustainable designers. Standard building practices do not use structural materials as efficiently as they could be used. Inefficient building forms that use excess concrete and steel contribute to the high environmental impact of the building industry as well as higher building costs. By taking structural loads through pure tension or compression instead of primarily through bending, less material can be used to span the same distances. These axial-load structures, along with reducing the amount of material needed for construction, are also visually appealing and have aesthetic value to architects and engineers alike.

However, standard building practices, such as rectilinear steel or concrete framed structures that use standardized sections are much more prevalent for a number of reasons. The first is that these systems are much easier for engineers to analyze. Repeated bays of beams and columns have known loads that are repeated over the course of the building. Once the geometry of the structural system becomes non-standard, the loads begin to change with changes in geometry. Whereas in a frame structure the section of steel or concrete can be made larger to take larger loads, once the geometry begins changing in response to loads, the loads themselves change, leading to an iterative, and more complicated design process. Also, engineers have been working with framed construction techniques for over a century; design, calculation, and

construction techniques have been developed specifically for this method of construction. Building codes are also developed with this system in mind, making deviation difficult.

Thus creating structures that are more efficient due to their shape is a challenge. Laying aside issues such as constructability and code restraints, a major problem facing architects and engineers is the design of these efficient structures, namely finding forms that can take the loads generated, and designing the structures to resist many different possible load cases. A traditional method of design for axial-force structures is creating physical models that use gravity to define axial force only shapes. Shapes formed by hanging chains require no bending stiffness, and are thus efficient geometries. Bringing this concept into the third dimension, a net of hanging chains also creates a shape requiring only tension forces to be held in the chains, since that is the only type of force a chain can take. When frozen and turned over, this hanging net creates an efficient shell shape, where the shell takes the load of gravity entirely in compression (Otto and Rasch 1995). Shells can take many forms, however this gravity shell shape is very efficient because no bending forces are induced under normal loading.

Structural engineers today have a wealth of different programs available to aid them in their work. Most of these programs are Finite Element Modeling (FEM) programs that analyze a structure whose geometry is known. However these programs cannot tell an engineer how to make the structure better from a design standpoint, only return back the performance of that particular design. With these programs, the various loading cases as well as other structural and architectural concerns are accounted for only after the preliminary design phase. This methodology can lead to concessions in the efficiency of the design. A way to consider these concerns in early design development would greatly improve these structures. Thus we face a need for programs to help determine which shape is best considering these engineering concerns,

to be analyzed in depth using FEM later. Early-stage design programs would greatly benefit the practice of structural engineering by allowing engineers and architects to explore forms together and easily create design iterations that are both efficient and aesthetically pleasing.

## 1.2 Problem Statement

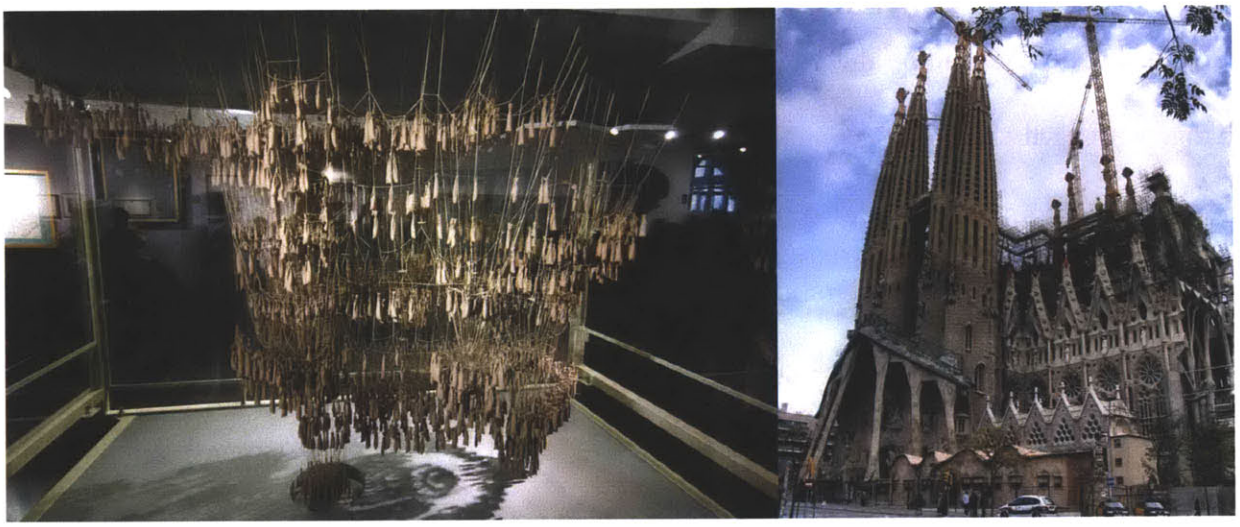
Architects have access to a plethora of design programs and methodologies of design with the introduction of parametric design and computational morphogenesis. However, very few of these techniques design with the forces imposed by gravity in mind. There is a great need for design programs that give the architect the creative power to determine the geometry of a project while allowing an engineer the ability to determine the way forces are transmitted through the structure in the conceptual stages of design. When looking at a design program specifically for axial force structures that are generated via the hanging method, the goal of interdisciplinary design can be achieved by having the architect create the initial geometry of the shell or net, and the engineer and architect together could then compute a design using the initial geometry. For early stage design, quick design iterations are also helpful for the design process. A designer should be able to make changes to parameters that affect the design and be able to see immediately the changes that occur in the geometry as well as changes in the forces felt in various areas of the structure.

Taking these goals together, the intent of this project as a whole is to fill a void in programs available to the design community for early stage design. Existing design programs do not achieve these goals as well as they could. This thesis expands upon the methods and individual programs used for early stage design of axial force structures through a new program, named *ForceDesigner*.

In deciding what should be included in the design program, a number of desirable functions were identified. The program should be specifically targeted at architects and engineers creating axial-force structures, since their design is as much controlled by the forces in the structure as their form. The program should be able to easily input initial geometry that has no design restrictions that can be generated separately in a different design session. It should quickly perform an analysis on the initial geometry and produce a funicular geometry that can be viewed on the screen. The user should then be able to change parameters of the global design and also of individual members, and be able to view the changes in real time. Since the forces in the structure change with each change in geometry, the program should also be able to return an estimate of the force in any member under simple loading conditions specified in the program. Finally, as part of the early stage design iteration process, the program should output the design in a useful format either for continued design refinements with the architect, or continued structural analysis using other programs. All of these goals were met in the final program design, and the following chapters will describe the project in detail.

## II. Literature Review

The problem of form finding for structural shapes is not new. A number of Twentieth Century designers used these concepts to design efficient forms. Spanish architect Antoni Gaudi (1852-1926) is famous for his complicated hanging chain models that led to rational and structurally efficient forms in masonry. An example of Gaudi's work is the Sagrada Familia, where he used strings weighted with sand to create the form (Schenk 2009). The model and the resulting cathedral are shown in Figure 2.1. German engineer and designer Frei Otto (1925-)



(a) (b)

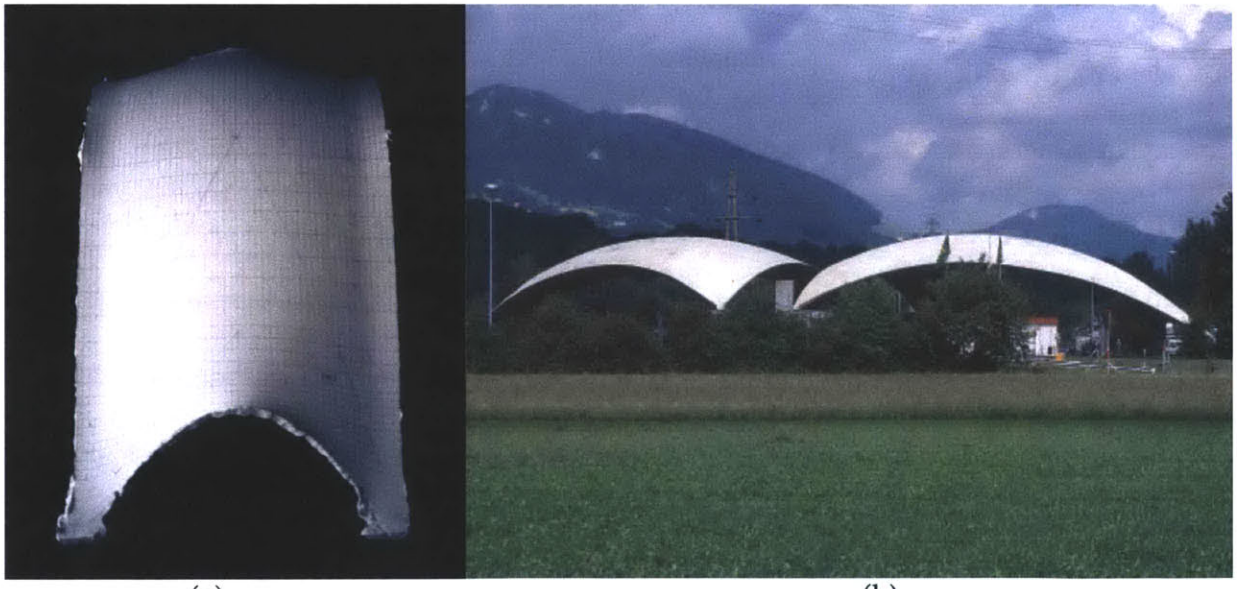
**Figure 2.1: Antoni Gaudi's Sagrada Familia. (a) Physical hanging model, replica created by Frei Otto. (b) Sagrada Familia under construction in Barcelona, Spain. (Schenk 2009)**

conducted research on hanging chain models and the forms that they are capable of producing at the Institute for Lightweight Structures. His team used these new methods of creating and measuring hanging chains to create a number of shells including the Mannheim Multihalle (1971), a shell created from discrete wooden laths shown in Figure 2.2 (Otto and Rasch 1995). Heinz Isler (1926-2009) used plaster-covered fabric to create hanging models, which would then

(a) (b)

**Figure 2.2: Mannheim Multihalle. (a) Physical hanging model (b) Constructed Multihalle (Block 2009)**

dry so they could be turned over and analyzed using a precision measuring jig (Figure 2.3 a). He used these models to create continuous concrete shells that spanned great distances while remaining proportionally thinner than an eggshell. The shells of the Deitingen Süd Service Station (1968), depicted in Figure 2.3 b, span over 100 feet with a thickness of less than 3.5 inches (Chilton 2010).



(a) (b)

**Figure 2.3: (a) One of Isler's hanging membrane models with measurement grid drawn on (b) Deitingen Sud Service Station, Flumenthal, Switzerland. (Chilton 2010)**

However, traditional physical modeling is time intensive and difficult to change, making design iterations expensive and inefficient. Taking measurements from these models is also time

consuming and difficult. John Chilton (2010) claims that this complexity is the reason more structures like this don't exist, and Isler's extreme precision allowed him to create such efficient structures. Digital technology has the potential to solve these problems with hanging models. Although a large variety of digital modeling tools and techniques exist, especially in the fields of analysis and optimization, few attempt to solve the problem of replicating hanging models for the problem of form finding.

The mathematical basis for creating hanging models has existed for some time. H.J Scheck devised the Force-Density method for form finding. This method involves creating an initial geometry and applying network analysis, creating a hanging net shape. The introduction of other parameters can create various other shapes (Scheck 1974). x be discussed more in Chapter 3. Although these methods exist, they still need to be available for designers to use. Few implementations of these methods exist for use by designers, although there is an important precedent to the work in this thesis.

The program CADenary, written by Axel Kilian is the first to successfully implement the particle spring system of structural form finding. The program utilizes the processing language to show the user how the structure hangs on the screen, and also allows the user to change aspects of the mesh while the structure is running (Kilian, Linking Hanging Chain Models to Fabrication 2004). It is one of the first a programs for structural design, not analysis, since it allows the designer to start from nothing and create axial-force only forms (Figure 2.4). There are a number of areas for improvement with CADenary, which this thesis will address. Firstly, although the user can make the geometry on the screen, the geometries are limited to either square meshes or linear springs. The program makes great progress in allowing users to create designs, but CADenary currently lacks any way to analyze the resulting structure, or to size the members in

different materials. Finally, the interface is difficult to master. This thesis project will use many of the same digital modeling techniques as CADenary, but will improve upon some of its lacking functionality.

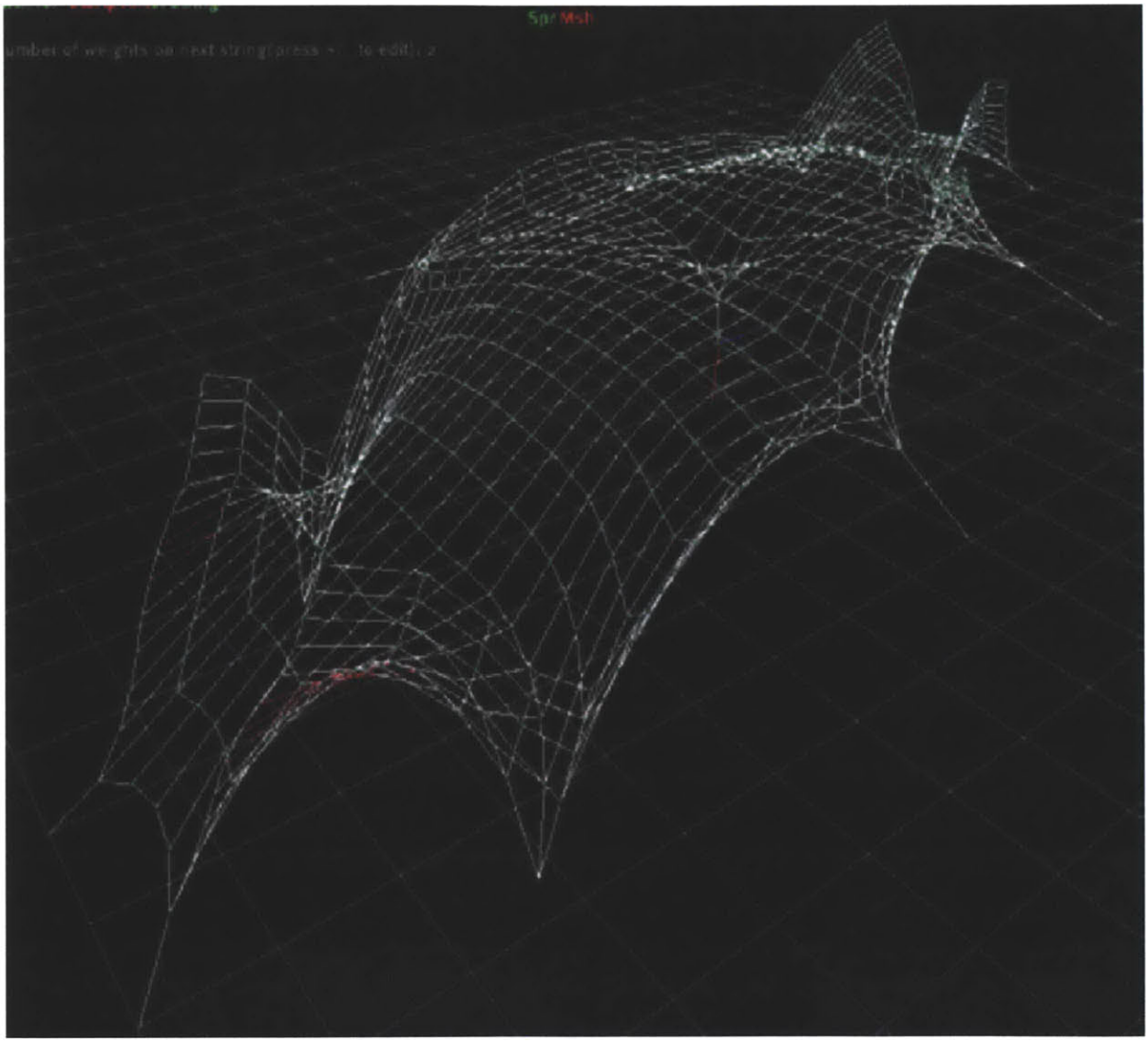


**Figure 2.4: Freeform shell created in CADenary (Kilian and Ochsendorf 2005)**

The next chapter will detail the methodology of creating a new design program for axial force structures, based on the precedents in this chapter and the design goals outlined in the problem statement.

# III. Methodology

## 3.1 Overview

This chapter outlines the methodology behind the design and implementation of the early stage design program outlined in the problem statement – named *ForceDesigner*. It will detail how the program works and how it improves upon its predecessors by implementing increased functionality. As outlined in the introduction above, the desired flow of the program is to have the user input any desired initial geometry, then be able to change parameters and see the designed shape on the screen in real time. The user should be able to have first-order force calculations for the solution as well as the ability to export it to other applications for further refinement. This flow is represented in Figure 3.1.

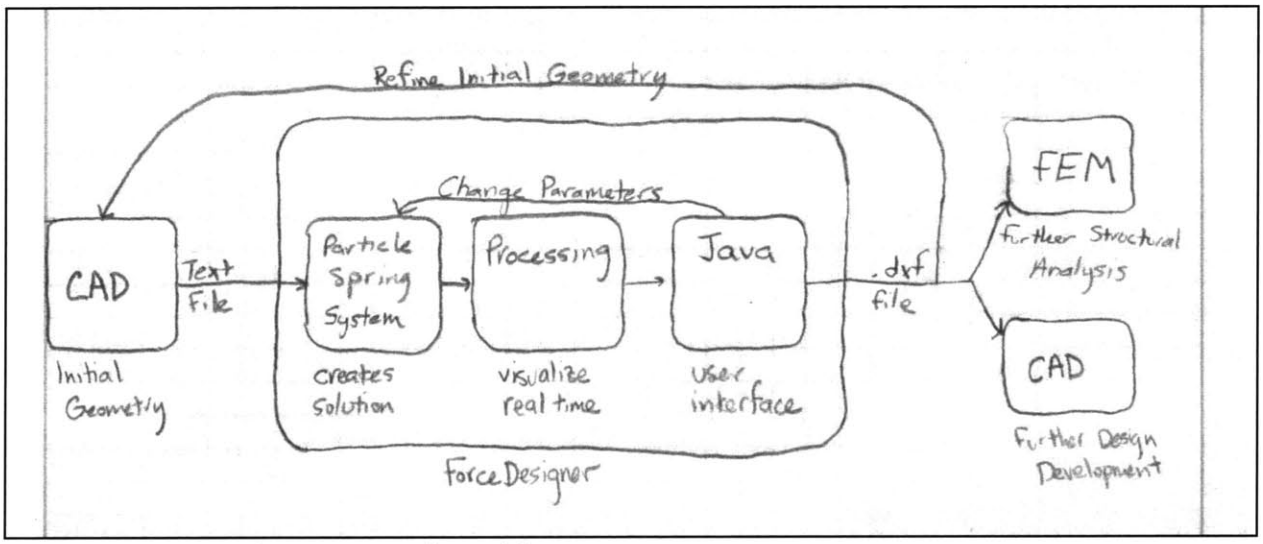



**Figure 3.1: Diagram of ForceDesigner's Workflow**

The following sections will discuss each of these steps and how they were implemented in *ForceDesigner*. First will be a discussion of the particle spring physics model that powers the design methodology as well as how it was implemented in the program. The next section will discuss how initial geometries are created and entered into the program. The following section

will detail how the Java and Processing programming languages allowed for increased functionality and improved user experience. The final section will be a description of a previous project that involved creating physical hanging models that can serve as a validation of this design program.

## 3.2 Physics Solver

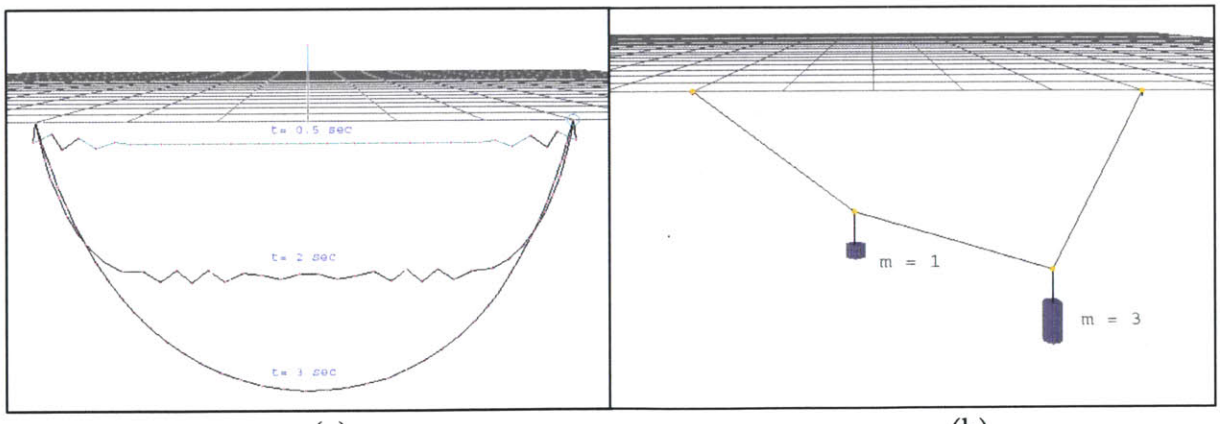
The goal of this design program is to easily replicate how a physical hanging model would generate a structural shape that has only axial force in its members. Thus the physics model driving the program must quickly derive a shape from an initial geometry that represents how a hanging model would act, while also being easily implementable in the flow of the program. As described in the previous chapter, a number of solvers have been developed that work well. The particle spring solver system was chosen for this project since it works well and a working particle spring model for Java had already been developed by Simon Greenwold, thus making implementation much easier (Kilian and Ochsendorf 2005). The beginning of this section will describe how the particle spring system works, while the second will describe how it was implemented and became the core of the *ForceDesigner* program.

### 3.2.1 Description of Particle Spring System

As its name implies, the particle spring system of structural form finding works by modeling a structure as a system of weighted nodes, called particles, connected by perfectly elastic springs. Each node of a structure where two or more members meet is represented by a particle. A particle has mass and a position and velocity in space. A member of a structure is represented by a linear perfectly elastic spring. A spring has an initial resting length, an

individual stiffness, or k-value, and is always connected to two nodes, where the connections have no moment capacity.

When released from its initial position, the weight of the nodes in the system will accelerate them downwards, stretching the springs from their resting lengths, which cause forces that are pulling the system back together. The system will move until it reaches equilibrium. The final positions of this equilibrium are dependent on the initial geometry and the k-values of the component springs. According to Hooke's law, F=kx, the spring will extend further when the k value is lower, assuming equal force on the springs. The process of reaching equilibrium, and the equilibrium position of a simple model is shown in two dimensions in Figure 3.2, but it works the same way in three dimensions. Although the final geometries may be different, they will always be axial force only, since neither the springs nor the connections have any moment capacity. The final shape can then be translated back into a structure with only axial-force members with connections at the nodes. The force in each member is also easy to calculate, as it is the spring's stiffness multiplied by its deviation from resting length. The particle spring system is described in much more detail in Kilian and Ochsendorf's paper on the topic (2005).




(a) (b)

**Figure 3.2: (a) Process of particle spring system reaching equilibrium (b) solution of simple system (Kilian and Ochsendorf 2005)**

A common misconception is that the structure resulting from the particle spring method of form finding must be able to be formed in the same way as the final geometry, and as such the members need to be just as elastic as the springs used in the model. This is not true, the final geometry can be built with materials with a much higher modulus of elasticity (such as wood, steel, or concrete), and these members will experience the same forces as the springs in the model. This fact is dependent upon the static determinacy of the particle spring system, or in the case that the system is not statically determinate, that all of the constructed members have the same modulus of elasticity. Beyond these cases, the force estimation ability of the particle spring method breaks down. Static indeterminacy will be discussed more in the next chapter.

### 3.2.2 Implementation in ForceDesigner

In order to carry out the form finding operation using the particle spring system, the system has to be written and coded in a manner that works well with the other aspects of the design program. Luckily, since this thesis expands on the works of others who created design programs using the particle spring model, a particle spring model already exists and is coded in the Java programming language. The library of Java code written by Simon Greenwold provides the core of *ForceDesigner*'s functionality (Greenwold n.d.). The library provides the ability to model a number of different things in a physical space, but this project used its ability to easily create a particle spring system that it can then solve in a stepwise fashion. The code takes a list of node objects and spring objects and loads them into a ParticleSystem, which is an object that keeps track of all of the particles and springs in space. It also applies gravity to the particles as well as applying the reactionary forces of the springs, and using an implicit Euler method of

solving differential equations, it can determine the velocity and position of each of the particles at any point in time.

Although Greenwold's library works very well for particle spring form finding, in order to add functionality that earlier design programs lacked, aspects of the library needed to change to better fit the overall program. One of the drawbacks of Greenwold's library as it stood was that every particle had the same mass with no method to change that mass. This did not change, since the goal of the program was an early stage design that usually consists of uniform loading over the surface of the structure. Thus equal masses at nodes was acceptable and changing that would have involved an in depth edit of the library. Another drawback was that the particles did not remember their initial position, which was necessary for editing the initial geometry from within the program. *ForceDesigner's* fixableParticle class is an extension of the library's Particle class that incorporates that ability. The last change is not fixing a drawback but adding functionality. *ForceDesigner* incorporates a coloring system to assign springs different k-values. The colorSpring class allows this by making the k-value assignment function of the library's Spring class dependent on what color the spring is. It also has a method to draw the springs that color on the screen and determine the force in a spring based on color, which is useful to adapt the geometry by assigning different k-values to different areas of the structure as well as understand the change in forces in these areas.

The library is also very well written in that it is self-contained. It requires the list of nodes and springs to be passed to it, with each node and spring having its initial data as described below. After that, it steps through the solution one step at a time, every time the draw method is called on the ParticleSystem. Although it is contained, the data is also accessible and easily altered. It allows the program to access the locations of all of the particles and springs and change strengths,

fixed nodes, and even the direction of gravity. This created a core that the rest of the program could be built around, which will be described in the rest of this chapter.

## 3.3 Initial Geometry

The particle spring system requires as inputs the location of the nodes and the endpoints of the springs to work correctly. It also needs to know which nodes are fixed location, which means they are serving as the structural supports and need to transmit a reaction force. Finally, in order to implement the coloring system of springs with different strengths, the model needs to know what color a spring is. These inputs are easily given to the model via a text file, which is how the program reads initial geometries. The input file is split into two portions: a list of nodes and a list of springs. Each node's entry consists of its node ID, its location in three dimensions, and a number that represents its fixity - 0 for free, 1 for fixed. Once that list is complete, the list of springs begins. Each spring's entry consists of its spring ID, the node ID of its two ends, and a number representing its color – 0 for black, 1 for red, 2 for blue, and 3 for green.

### 3.3.1 Generation of Initial Geometry for *ForceDesigner*

Using this system, a user could code an initial geometry by hand. However, that would be a long and error prone process. As the intention of *ForceDesigner* is to rapidly create geometry, an easy method of creating this input file was necessary. Using a CAD program to create the initial geometries would be much easier, especially since CAD has exact control and mass replication functions that allow the user to create vast grids easily. However, a CAD program with the ability to export to this text file format was also necessary. Rhinoceros 3D, a CAD program by McNeel and Associates, has the ability to write scripts, including functions that can output to text files (McNeel and Associates 2010). As part of an earlier independent study, a

script written for Rhinoceros 3D using the Monkey scripting environment was developed by Duks Koschitz, Ph.D. Candidate at MIT. This thesis adapts it for use with *ForceDesigner*, and the script can be found along with the *ForceDesigner* program files (Figure 3.3 a).

Running the script is very simple: the user has to select all of the line segments that will become springs (Figure 3.3 b), and then select points that will become fixed nodes (Figure 3.3 c). These nodes should coincide with the end of at least one line segment, or it will not work properly. The script creates the text file essentially in reverse. It first identifies every line segment selected and then identifies every endpoint and records its location as a node. If the end point was one of the selected points, it records the node as fixed. It then goes back through the line segments and records which nodes are its endpoints. It checks the color value assigned to each segment and records whether a segment is red, blue, green, or otherwise black. Finally it writes the text to a file, which can then be used as the input file in ForceDesigner (Figure 3.3 d).

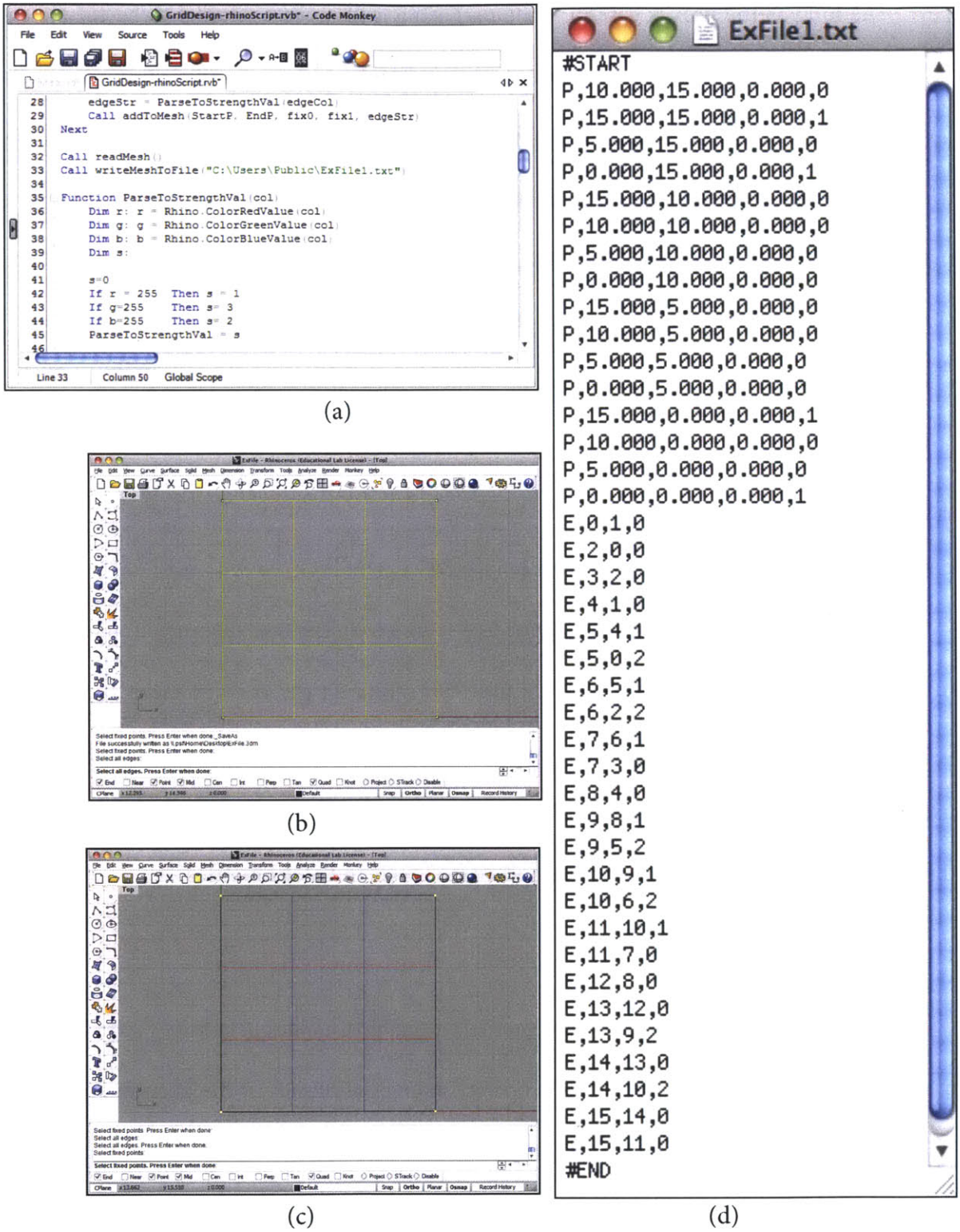



**Figure 3.3: Process of exporting initial geometry from Rhinoceros. (a) MonkeyScript that converts geometry (b) Selecting all members on screen (c) Selecting fixed points on screen (d) Final text file**

As long as the user is careful while preparing a Rhinoceros model to be turned into an initial geometry, this is a very robust way to input an initial geometry of almost any scope. The largest problem is springs not connecting together, but as long as line segments that are intended to be connected share endpoints, and long segments with other segments intersecting in the middle are broken into smaller segments, that should not be a problem. The program tends to slow down considerably with geometries that are large scale or that have greater than 5,000 nodes and segments. However, due to the nature of the particle spring model and the fact that spring stiffnesses are adjusted for resting length, an identical geometry at any scale will produce an identical result. This means that there is no need to create models to exact scale to produce valid results; the results will be valid at any scale.

Although the method using Rhinoceros works well, the text file format allows flexibility in how initial geometries are created. If a user can create scripts or some other means that will translate geometries in any other CAD program into a text file with this same format, it can be used with *ForceDesigner*. Thus as CAD evolves and different programs become more popular, this program will not be tied to only Rhinoceros to create initial geometries.

### 3.4 Visual Rendering

Once the program has the initial geometry, the next major goal of the program is that the user should see gravity pull the initial geometry using the physics of the particle spring system. The method of visual rendering of the geometry needs to work with the particle spring system so that the movement of the system can be seen in real time. Other useful features for the visual rendering system include a camera that can zoom, pan, and rotate in three dimensions and the ability to specify exactly how each member and node will be rendered, including color and size.

### 3.4.1 Processing

A rendering system that fits all of these needs is Processing. Processing is a programming language that was created in the MIT Media Lab and specifically developed for visualization and animation uses (Fry and Reas 2004). The language is built on Java and uses similar conventions, but it makes creating shapes and animations on the screen much easier than using the Java language. CADenary also used Processing to render digital structural models, and the particle spring library was written to work with Processing so the decision to use the programming language was natural (Kilian and Ochsendorf 2005). Processing also comes with its own IDE, or integrated design environment. This allows code to be written easily and most of the detailed work in getting different sections of code to work together is already done for the user. However, since *ForceDesigner* uses Processing as one piece of a larger program, the difficult task of integrating the Processing code into the larger Java code had to be done in a different IDE, which will be discussed in the next section. For more information on the Processing language and IDE, refer to its website (Fry and Reas 2004).

### 3.4.2 Using Processing

Processing code is generally split into two blocks, the setup and the draw. The setup block defines what is on the screen at the beginning and any other objects or methods that need to be in place before the program starts running. After that is complete, Processing will run the draw block continuously for as long as the program is running. *ForceDesigner* utilizes this by importing the initial geometry and initializing the particle spring system in the setup block, and then having the physics system run through one step and display it on the screen every time the draw block runs. Thus the user can see the initial geometry fall into the final geometry, as the

program steps through multiple times a second. This progression can be seen in the set of stills that comprise Figure 3.4. Also, since the particle spring system is accessible to outside code while the program is running, if the user changes any of the properties of the model, the effect is seen immediately in the visualization since the draw block is continuously running.

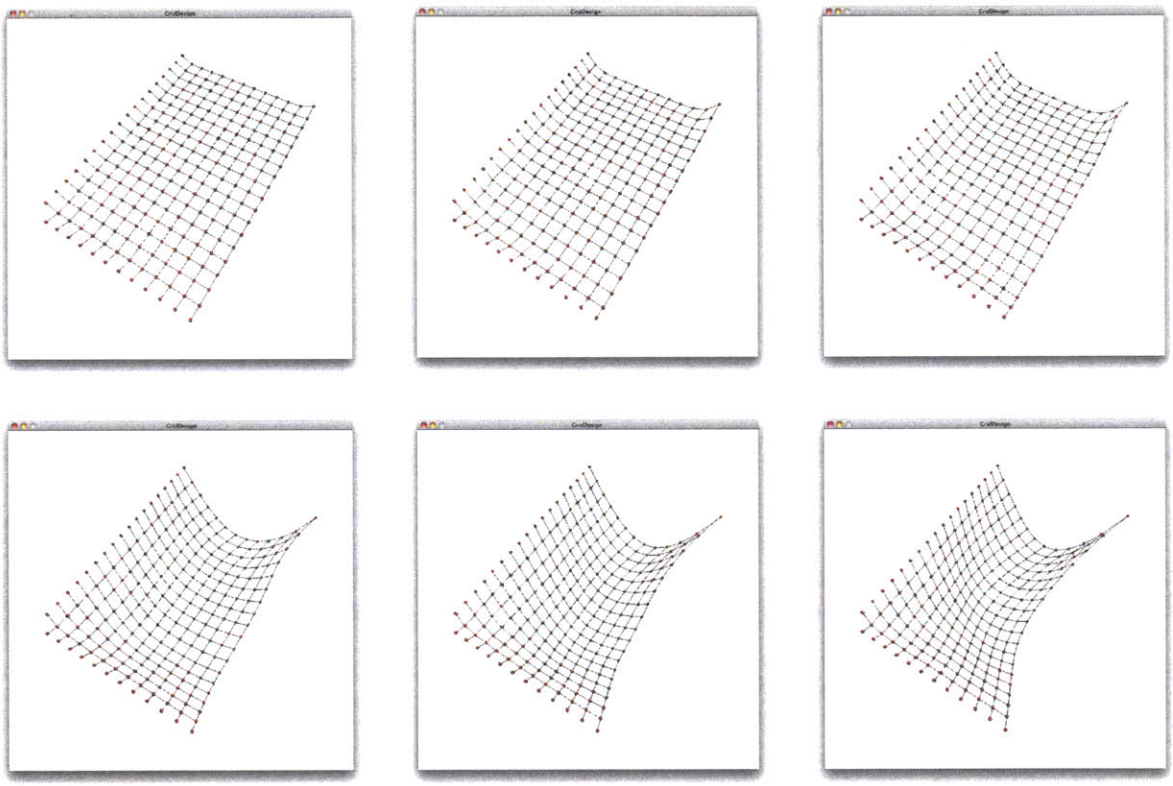


**Figure 3.4: A series of stills showing Processing's rendering of an initial geometry finding its equilibrium position**

Processing also has the ability to easily import code written by others in the form of libraries. Along with Simon Greenwold's library for the particle system, *ForceDesigner* uses a popular Processing library called PeasyCam to allow the user to view the model from any angle or zoom level (Feinberg 2008). *ForceDesigner* also uses the DXFwrite library that easily exports the final geometry back into a DXF format for use with other design programs (Fry and Reas 2004). *ForceDesigner* uses Processing for the main model view and for the editor view that shows the initial geometry in plan view, but allows the user to pan around and zoom in two dimensions.

Since Processing is continuously running the same code, it does not work well for other aspects of the program's functioning. As a remnant of its ease of use for visuals, it does not work well for having multiple windows or having an easily implementable user interface for changing model properties. Thus one of the major innovations of *ForceDesigner* over earlier programs is combining the visualization power of Processing with the organization and user interface abilities of Java, described next.

## 3.5 Program Structure and User Interface

Many of the novel functions of *ForceDesigner* come in the user interface and the ability to change the design from within the program. Thus the Java language needed to be used to enhance the power of the program and allow facets of the model to be changed from a graphical user interface. Using Java also allowed for multiple windows that can interact with each other and a more professional looking product.

Considering the desired functions of the program, the goal was to have four panels that would make up the interface. Fig. 3.5 is a sketch of the desired interface components. The components consist of one that would be a general control window, one that would display the model, one that would control the editing functions, and one that would control the engineering outputs. The output view panel was essentially already written from earlier work using the particle spring library in Processing, but the others had to be newly designed.

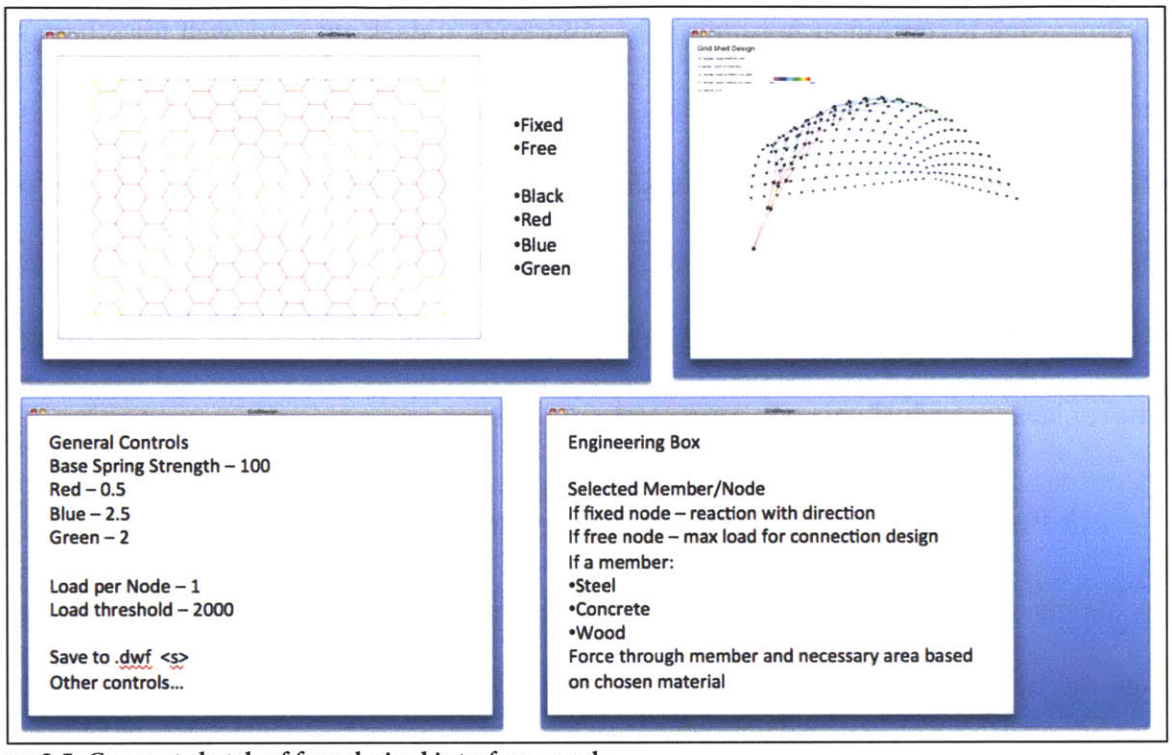



**Figure 3.5: Concept sketch of four desired interface panels**

The graphical user interface is implemented using the Swing package for Java. A common organization scheme for using swing is the Controller-Model-View structure. The model class can make decisions and make computations based on what the user does, the controller class is what passes information from the user to the model and also from the model to the user, and the view class interprets model data to show shapes on the screen. In this case, the Processing code that makes up the Swing model class also serves as the view class, since it has the built in graphics capabilities. Thus, no view classes were necessary. Since *ForceDesigner* has multiple panels, multiple iterations of this structure would be necessary, which in itself needed an overarching structure. The control panel is the main controller, and it contains buttons to open or close the other panels. The model linked to the control panel is in fact the 3 dimensional view, which is written in Processing and contains the particle-spring code. In order for everything to work together, the processing view was integrated as a panel in the larger view window.

The editor pane is controlled by the control panel, and also contains a separate Processing class that controls the plan view that can be interacted with. It also must have a separate class that interprets the commands to change the model and update the controlling class above to change the particle spring model. Finally the force calculator does not need to control any parts of the model, but must access the model in order to return force values that it must do further calculations with. Thus it has its own model class that does these calculations and pulls data from the higher class, and a controller that allows the user to specify the engineering properties he is looking for.

All of these classes are brought together in one Java package, which can be run without needing to interact with the code below. The Swing package also standardizes how the interface will look across platforms, and also allows any input file to be put in and to output to any desired output path. The program can also be easily hosted online or distributed as well as edited to work better, look better, or add more functions. The final appearance, functionality, and intended uses of *ForceDesigner* will be discussed in the next chapter.

# IV. Results

The result of the work coding in Processing and Java is the *ForceDesigner* program, whose usage will be described in the next section. After that will be a discussion of what sorts of initial geometries provide better results after being run thorough *ForceDesigner*.

## 4.1 Usage of Force Designer

After creating a text file of the initial geometry as described in chapter III, the user can begin using the program. Upon opening, the program prompts the user to enter the path of the initial geometry text file, as shown in Figure 4.1. If it resides in the folder that the Java project also resides in, all that is necessary is the name and the .txt suffix.

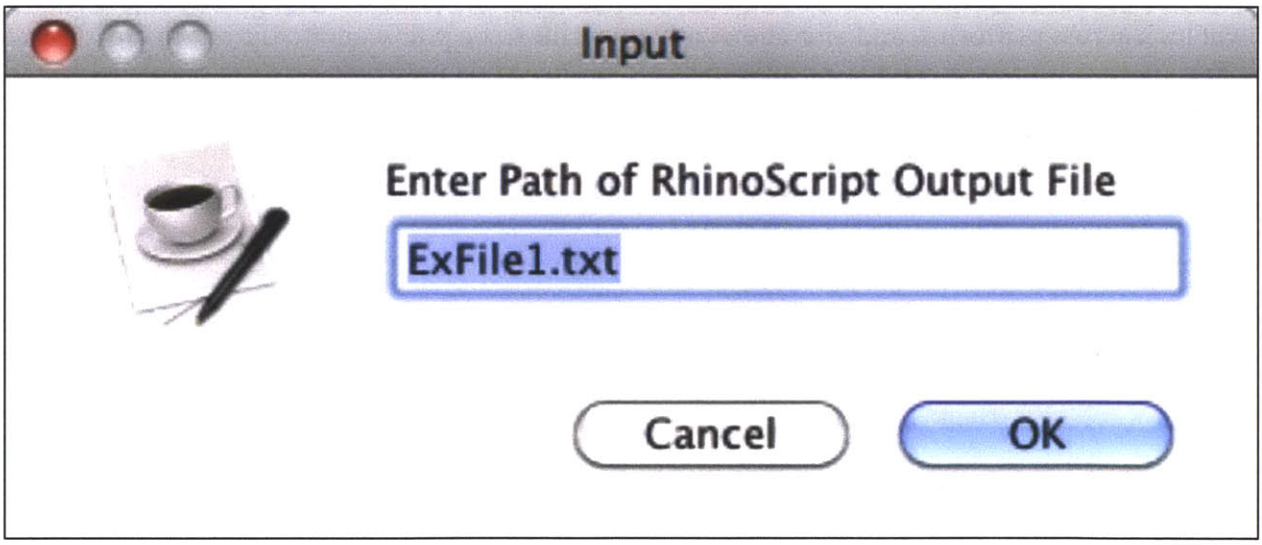



**Figure 4.1: Prompt to input initial geometry file**

Upon successful loading of the initial geometry file, the program will open with two windows open, the control panel and the design view, as shown in Figure 4.2. The view will immediately begin to change, as gravity is applied to the initial geometry from the beginning with

an initial set of parameters. The next four sections will describe the functions on these windows and the two others that are accessible from the control panel.

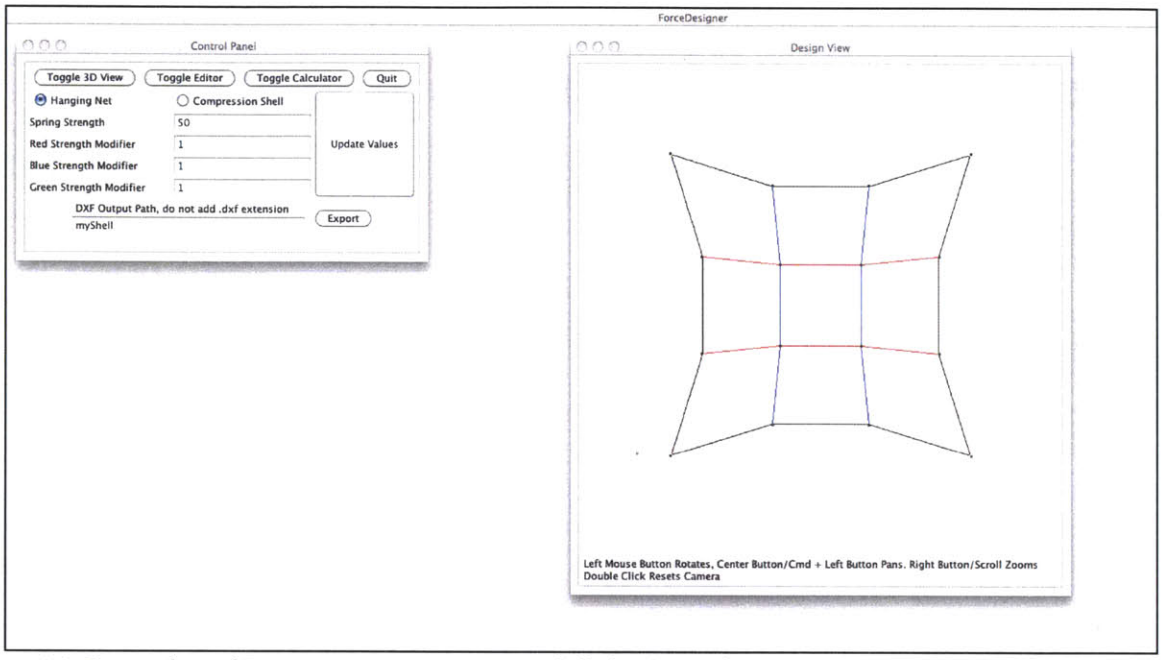



**Figure 4.2: Screenshot of ForceDesigner upon successfully loading a file**

### 4.1.1 Design View Panel

The design view panel shows the current state of the model. The particle spring system is always running while the program is on, so the nodes and springs are consistently trying to find an equilibrium position. The view panel has a camera associated that is capable of zooming, rotating, and panning, all by using the mouse. The various camera angles are shown in Figure 4.3. The instructions for using the camera are printed on the panel itself. By double clicking the mouse, the camera will return to its initial position, which is a plan view of the geometry. Whenever a change is made to the model in any of the other panels, it will automatically be reflected in the view panel. It may take a few moments for the model to find its equilibrium position again after a change is made, but the user can see the model moving and determine

when it has reached equilibrium. Once that happens, the engineering data produced will be accurate and the geometry can be exported accurately.

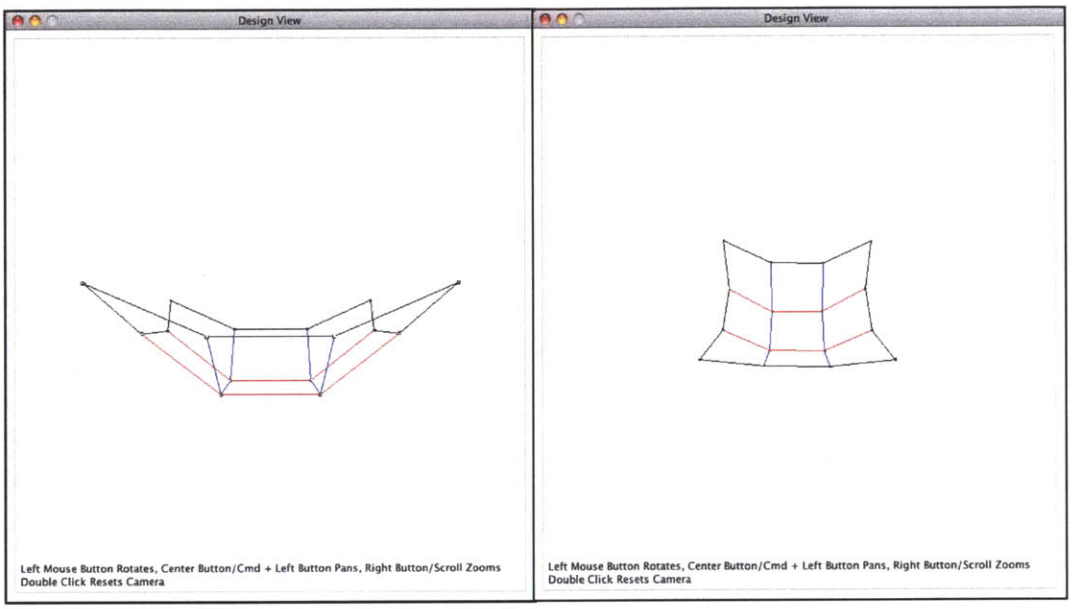



**Figure 4.3: Screenshot of various possible camera angles**

### 4.1.2 Control Panel

The control panel is where the user controls the functions of the program as well as many of the parameters of the model. When first opened, it looks like Figure 4.4. The top row of buttons toggles the other three windows of the program. The control panel is the only window that must be open at all times. The final button in the top row quits out of the program.

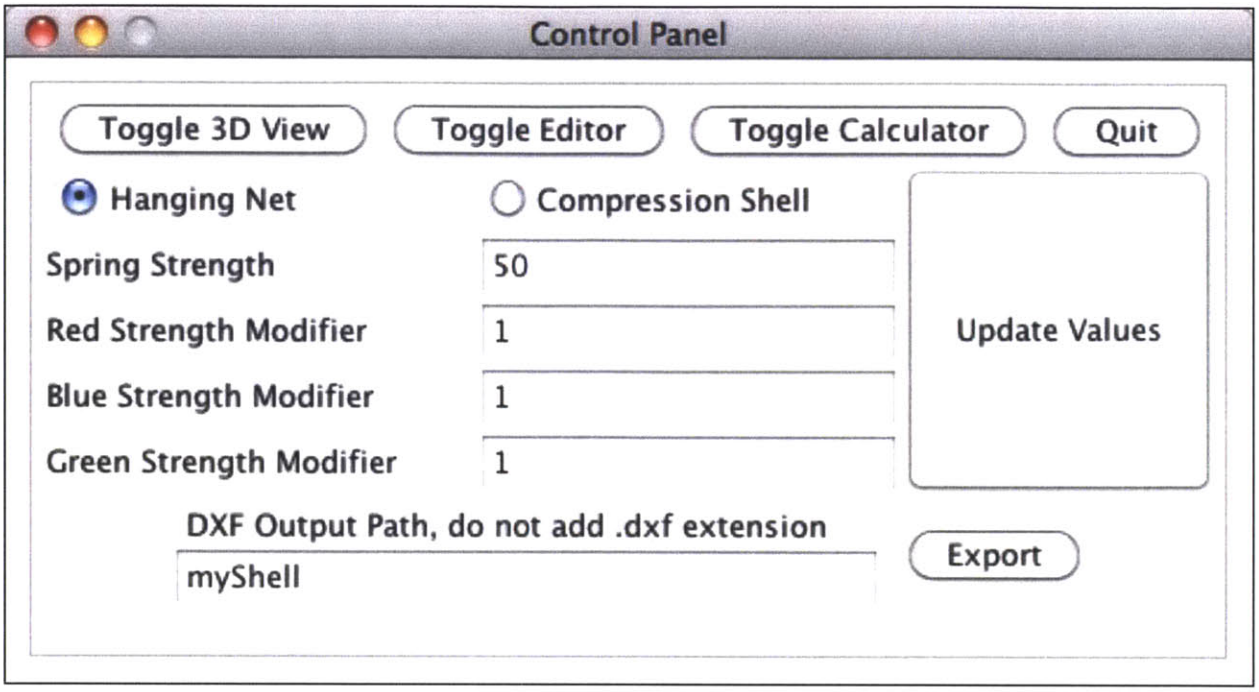



**Figure 4.4: Initial state of the Control Panel**

The next set of radio buttons controls the direction that the model wants to hang, and thus whether the springs represent tension members or compression members. The program can be used both for tension membrane structures as well as compression shells, and the user can switch between them by choosing a radio button and hitting the Update Values button. Although when the initial geometry is flat this direction does not make a difference, as the model can just be inverted, but when the initial geometry involves different elevations this is necessary for creating the correct shape. The difference between Hanging Net and Compression Shell are shown in Figure 4.5.

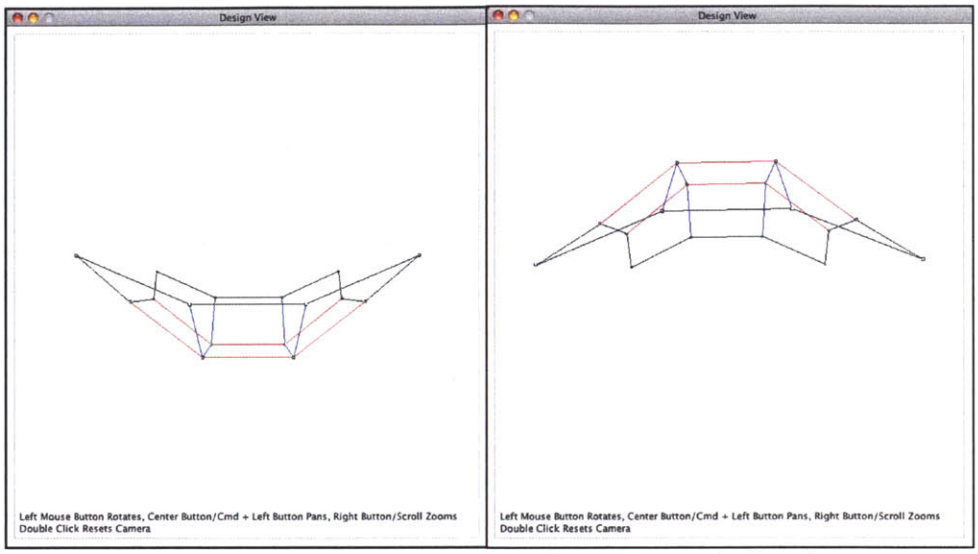



**Figure 4.5:Comparison of identical models in hanging net and compression shell mode**

The next four lines in the control panel allow the user to control the k-value, labeled as Spring Strength. The first box is the overall spring stiffness of all of the springs in the system. The initial value is 50, and any value that the user inputs is relative to that. The ratio of the number entered to 50 is the ratio of the new extension past resting length to the original. Thus by entering a value of 25, the springs will *extend* double as much as they will at 50, and by entering 100 the springs will extend half as much. Very high and very low numbers tend to crash the program; there is no way to make the spring extension zero. The next three lines are the relative strengths of the three possible colors, red, blue, and green, compared to the base spring strengths. A value of 1 means that color is the same strength as the base value, a value of 2 means the spring strength is double the base value, and so on. These changes take effect all at once upon the Update Values button being clicked.

The last function on the control panel is the ability to export the current geometry to a drawing exchange file, or *.dxf*. The user inputs a file name without the *.dxf* extension, and upon clicking export, the current state of the model view panel will be written to a file and the file will

reside in the project folder. It is important to remember the file will be written of the state of the model at exactly the time export is clicked, including any camera angles, so it is advisable to home the camera before exporting. There is otherwise no change to the model when exporting as a file.

### 4.1.3 Editor Panel

The user can reach the Color and Fixity Editor from the control panel (Figure 4.6). The top half is a plan view of the initial geometry with numbers overlain. These numbers are the IDs of the springs in the model. These are necessary for any editing operation. The user can pan and zoom around the initial geometry to find spring IDs, or the user can click the toggle to turn off the numbers.

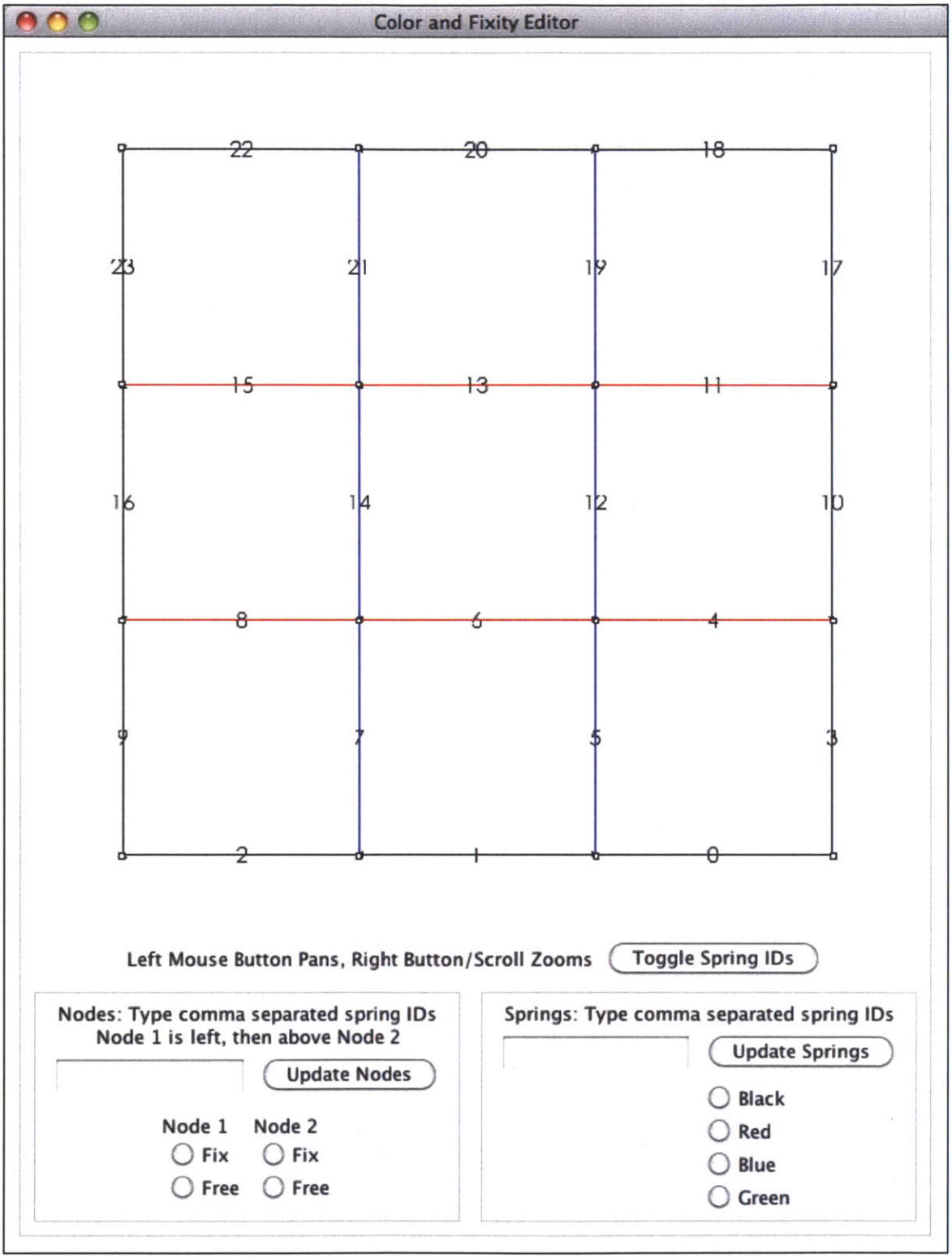



**Figure 4.6: Color and Fixity Editor**

Below, on the left side, the user can change the fixities of nodes. The user enters a spring ID or a comma separated list of IDs, and chooses fixed or free for the node on either end. The particular node represented by Node 1 and Node 2 is explained on the panel. After clicking Update Nodes, the model instantly updates with the desired changes. Fixing a node fixes it at its *initial* position, not at its current position. Freeing a node allow it to fall and be acted on by its surrounding springs. A node can be fixed or freed by any of its attached springs, so the user should attempt to avoid contradictions by fixing and freeing a particular node at the same time.

The right side allows the user to change the color of springs. The user must enter a spring ID or comma separated list of spring IDs and choose a color, including the standard black. Then by clicking Update Springs the listed springs will change color in all of the views and instantly take on the strength properties of that color. These functions will be explored further in the design examples in the next chapter.

### 4.1.4 Force Calculator Panel

The fourth panel, also accessible from the control panel, is the Force Calculator. It allows the user to determine loads in the members represented by the springs and reactions at nodes. At the top, the user needs to enter the load each node represents. This is typically the tributary area of each mesh unit multiplied by the expected loading on the surface. As this is only used as an early stage design program, the loading options are limited to a uniform load on every node. Then the user needs to choose a material for the proposed member, which will determine how much area is needed of that material. The user can pick from common materials or enter any allowable stress value to determine the area needed for that member. Finally the user must enter

a single spring ID from the editor panel to analyze. Upon clicking Calculate, the panel will refresh with number values in the bottom half, as shown in Figure 4.7.

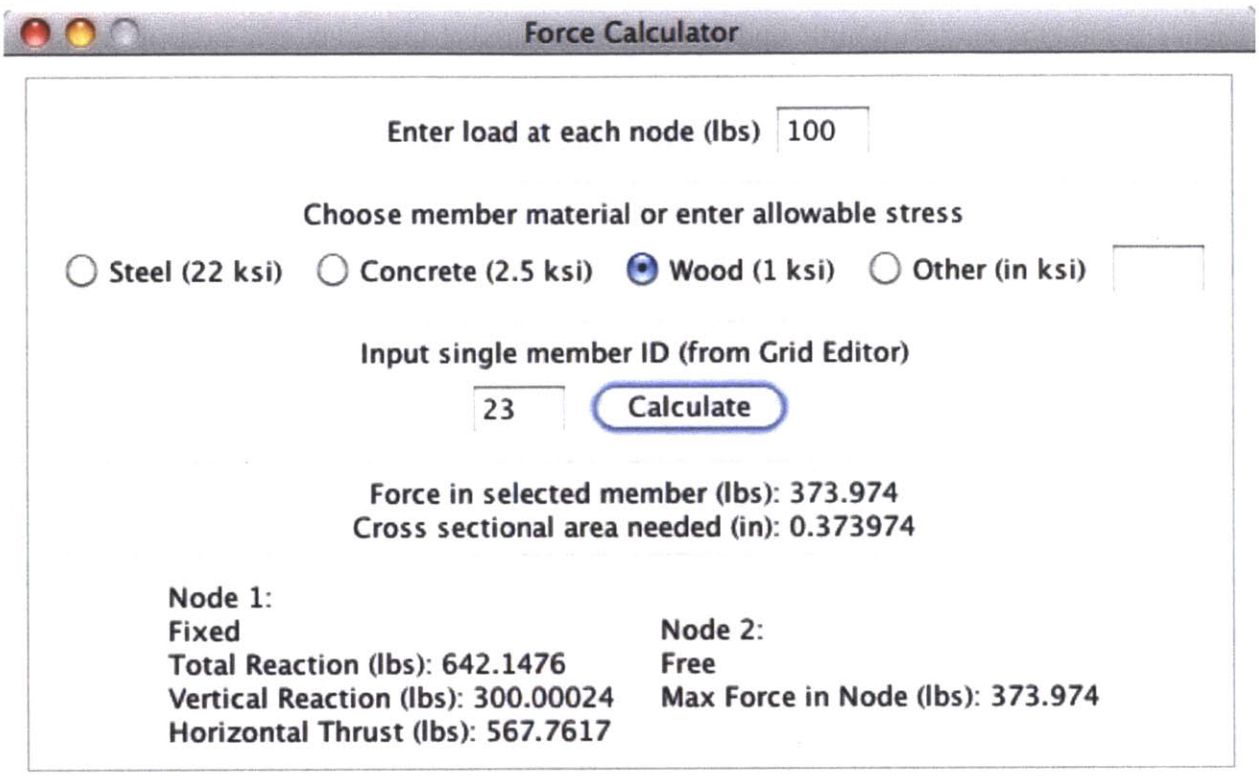



**Figure 4.7: Force Calculator**

The first pair of numbers is the load in the member represented by the selected spring, and the second number is the area needed based on the selected material. Below that, the panel shows results for the two nodes attached to the spring. If the node is free, it shows the largest force going into that node from any spring connected to it, which is useful for preliminary connection calculations. If the node is fixed, the panel shows the total reaction force needed to hold the fixed node in place. The panel also separates the reaction into vertical reaction and horizontal thrust values. These are helpful for first order foundation designs and other reaction calculations.

## 4.2 Guidelines for designing initial geometries

Although the program can handle any of initial geometry, for the model to work as well as possible the designer should follow some guidelines. The node density of the initial geometry represents the load density. There is no way to change how much any node weighs, so having sections with more nodes represent areas in the final structure that will be more heavily loaded, which for most early stage designs is not desirable.

Along with node density, the spring lengths matter. The amount of stretch is a function of resting length as well as spring strength. If the model contains two springs with strength of 50, one with a resting length of 3 and another of 6, the longer one will stretch double. This can lead to issues if trying to mesh with different length springs. It can also create problems when adding anomalies into existing meshes.

### 4.2.1 Static Determinacy

The particle spring system runs on the assumption that all springs will always be extended. Cases where this may not happen are undesirable and may cause the springs to ripple and create folds in a model, or return invalid engineering values and forms. A way to make sure this does not happen is to create statically determinate meshes. In three dimensions, a node is statically determinate if it has three or fewer springs attached. A statically determinate node has only one load path that leaves the node motionless. If a node has two springs, it cannot take forces in all three dimensions, but can still be a part of an otherwise determinate geometry. If a node has four or more springs, there can be multiple load paths, and not all the springs need to be used to take the load. This can lead to springs with zero load or springs in compression. Thus, although

square meshes are the easiest to generate, they are less desirable than hexagonal meshes for modeling a continuous surface.

A second issue with determinacy is that even if an indeterminate mesh does not have any springs in compression, when translated into a physical structure the load path may not be the same. A determinate structure has the same load path no matter what material it is made of, or if different members have different elasticity properties. An indeterminate structure's load path changes based on the elasticity and strength of its component members. This means that the engineering values and axial-force geometry determined by the particle spring system may not be the same when built using real materials.

Now that the program and its functionality have been examined in detail, the next chapter will show some of its capabilities via design examples.

# V. Design Examples

## 5.1 Overview

In order to exhibit the abilities of *ForceDesigner*, three different design examples using the software will be shown. All three use the same methodology and functions of the program, but with different initial configurations and a variety of final designs. Each will also have a description of how it was created and how the techniques can be used for other designs, as well as pictures of the initial and final geometries.

The first example is a simple vault design that will show the basic functionality of the program for grid shell designs. The second example is a design replicating an existing building – Felix Candela's Los Manantiales Restaurant in Xochimilco, Mexico. The grid is more representative of a continuous shell instead of a discrete grid structure. The final example is a more open-ended design problem that shows the power of the structure to create multiple solutions in a short amount of time.

## 5.2 Vaults

In the first example, a designer wants to span 100 feet using a barrel vault. The designer would like the lowest height for the vault, but has a maximum thrust limit on the foundation. The vault must carry 60 lbs. per square foot of plan to account for the weight of the vault and snow loads. The vault will also be a discrete grid shell, so the designer uses Rhinoceros to create a flat grid of hexagons that will span the 100' distance, as seen in Figure 5.1. As discussed in Chapter IV, a hexagonal grid creates a statically determinate system with only one solution of forces. The designer then places fixed points at either end of the vault.

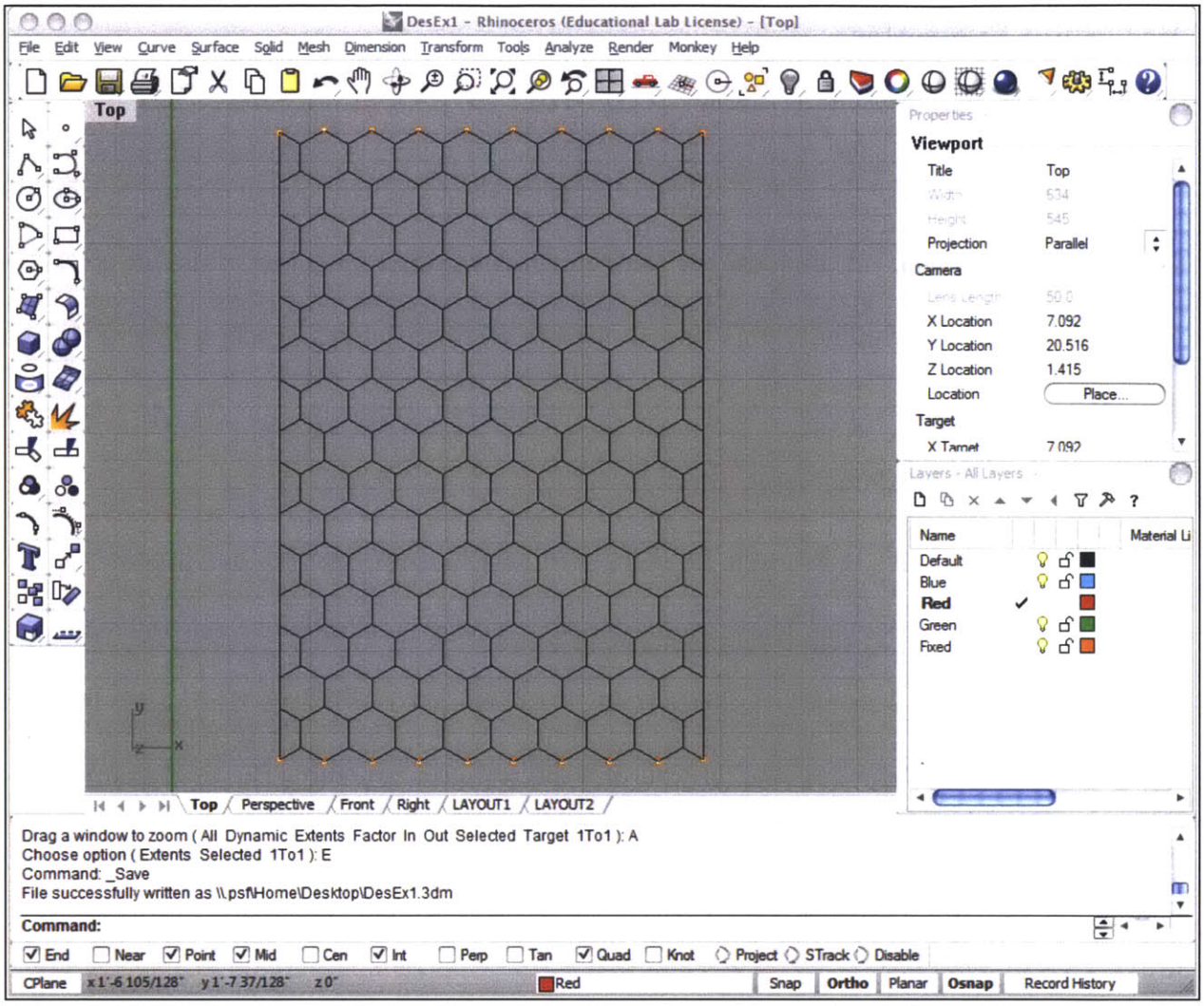
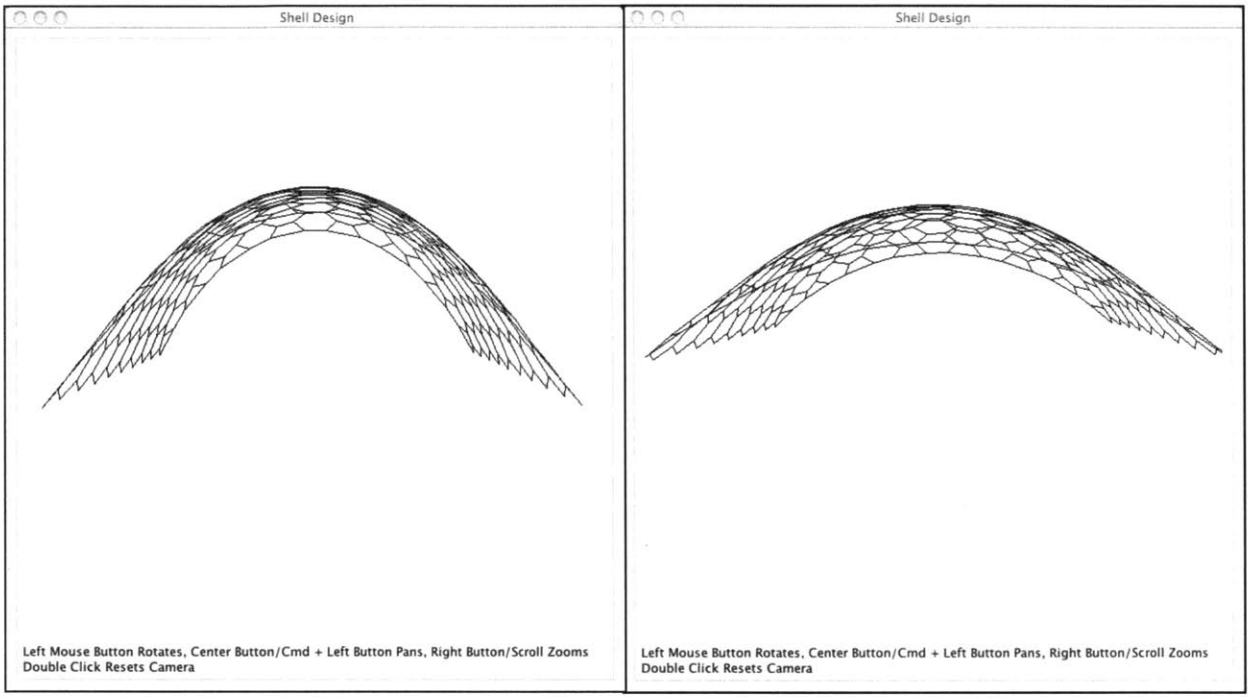



**Figure 5.1: Initial geometry for barrel vault**

After running the script that turns the model into a text file and importing the file into ForceDesigner, the vault looks too tall with the initial k-value of 50. In order to decrease the height, the designer increases the Spring Strength to 250 and is more satisfied with the height to span ratio (Fig 5.2).




(a) (b)

**Figure 5.2: Views of barrel vault with Spring Strengths of (a) 50 and (b) 250**

However, the designer wants to use the force calculator to determine the reactions on the foundation. Since the tributary area around one node is a triangle of 24 plan square feet (determined in Rhino), the load at each node is roughly 60 psi x 24 $ft^2$ = 1440 lbs. To check the reaction at one of the fixed foundation points, the designer enters the ID of a member next to a fixed node near the center, such as node 80, and as can be seen in Figure 5.3, the reaction is 19 kips with a thrust component of over 10 kips, which is very high for the specifications of the site.

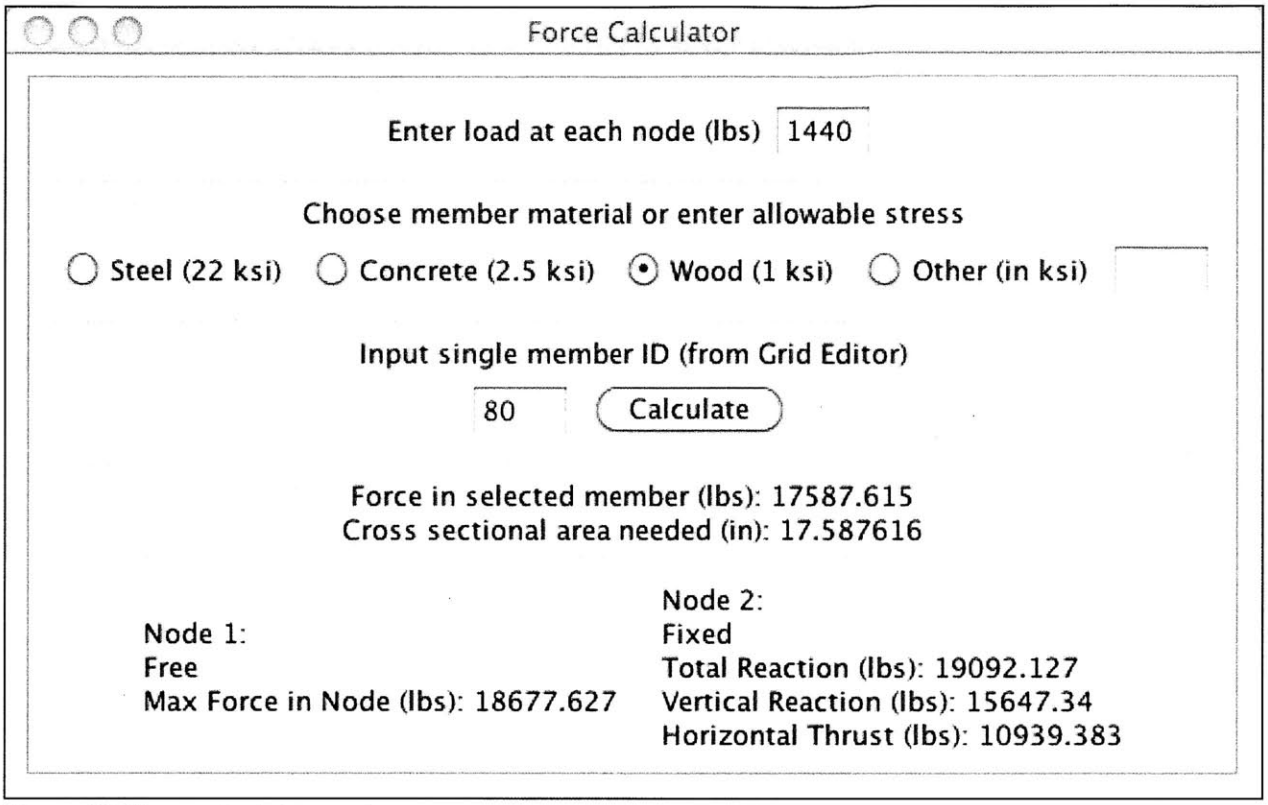



**Figure 5.3: Force calculator results**

As another option, the designer decides to see what a double barrel vault will look like. This is easy to accomplish by fixing the nodes of the center row of members using the Color and Fixity editor. Putting the k-value back at 50, this provides a solution where the height is not too much but the thrust is also reduced, at 6.5 kips (Figure 5.4).

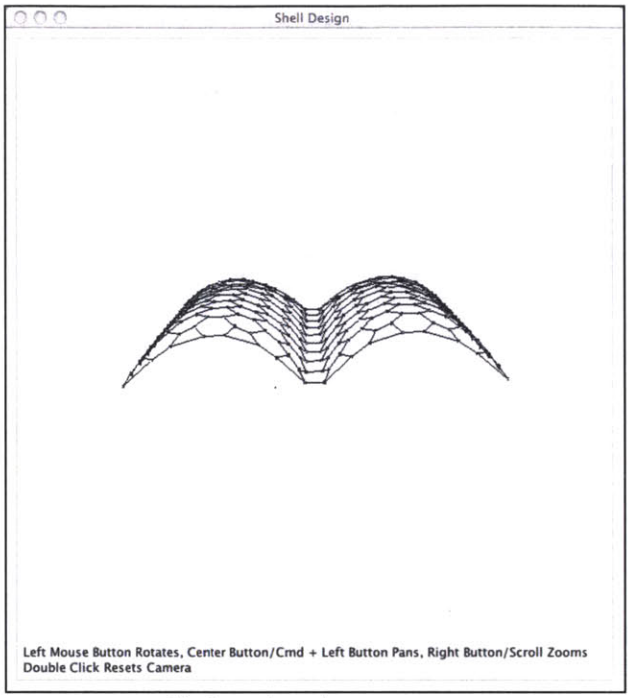



**Figure 5.4: Double barrel vault**

### 5.1.1 Ribbed Vault

The designer then decides to create a ribbed vault, instead of a barrel vault. This requires areas of the mesh with a higher spring constant than others. The designer chooses to use the exact same starting geometry as before with different fixed points. The color-coding functionality of the program is used to create these areas of different stiffnesses (Figure 5.5).

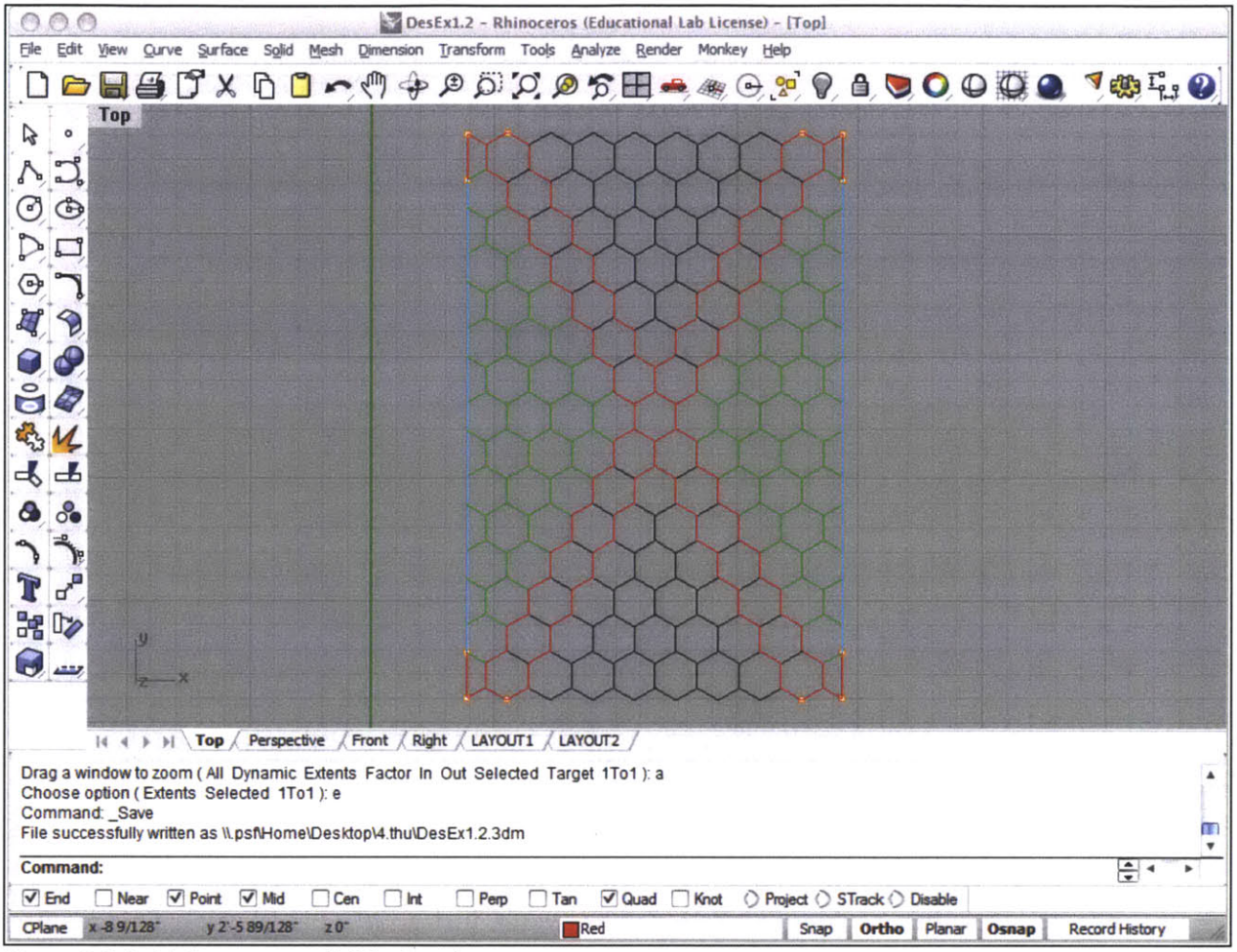



**Figure 5.5: Initial geometry of ribbed vault**

After loading in the geometry, the initial shape has all the colors at the same strength, giving a tall vault. The designer begins by increasing the strength of the red color, creating the ribs of the vault. However, the designer wants the opposite quadrants to also be of different strength, so that the short edge is higher than the long edge. Changing the green and blue strengths results in a ribbed vault, as the control panel and design view show in Figure 5.6.

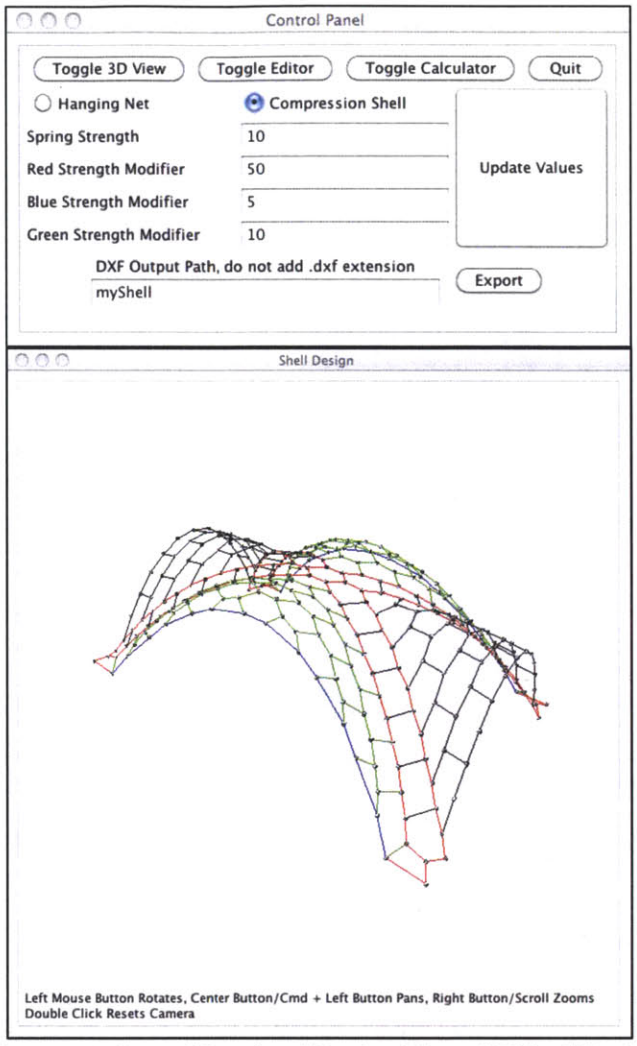



**Figure 5.6: Control Panel and Design View of Ribbed Vault**

The vault is very similar to the ribbed and groined vaults present in gothic cathedrals. In a ribbed vault, the ribs take a large portion of the compression stress and direct it to the foundation. This was important in cathedrals so that the load could be directed through the large stone ribs to the columns. This frees the other portions of the vault to be lighter (Figure 5.7). Just like in the cathedral, the sections of higher strength in this vault attract more force. The force calculator verifies that the red springs have the highest force, and the green springs in general have higher force than the black springs. All of this was done within the program using the same initial mesh,

and any of these designs can be exported to .dxf format to be used for further design development.

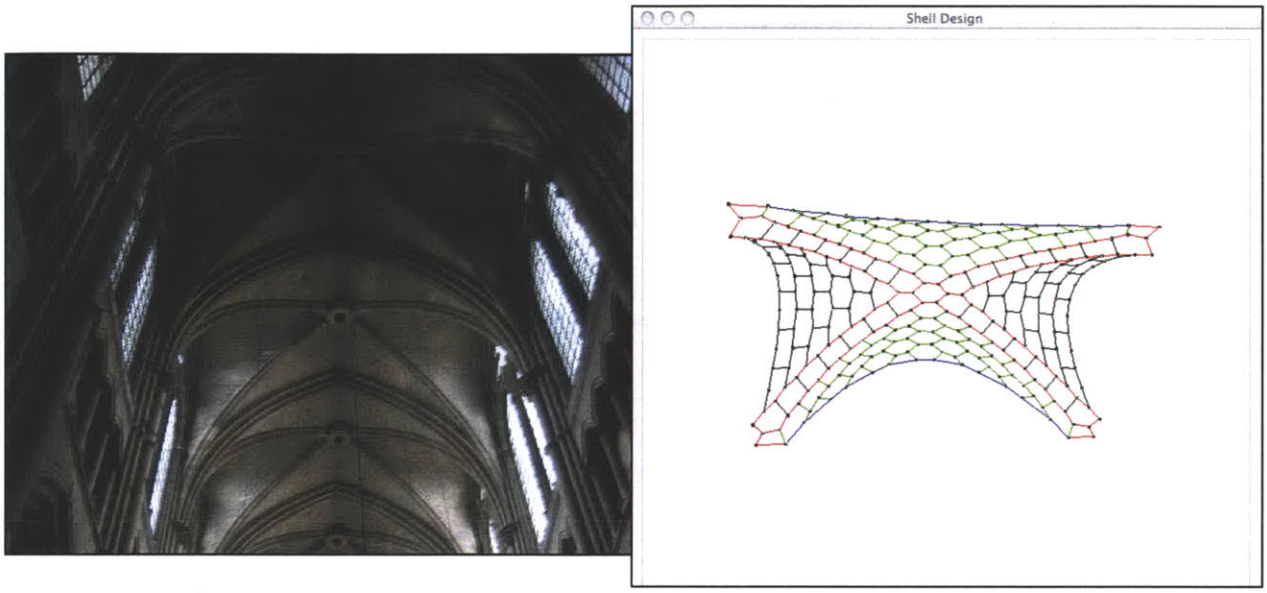



**Figure 5.7: Comparison of Chartres Cathedral vaulting to the ribbed vault created by *ForceDesigner***

## 5.3 Los Manantiales

In the second example, the designer wants to replicate as closely as possible the hypar shape of the shell roof the Los Manantiales restaurant designed by Felix Candela in 1958. The shape, shown in Figure 5.8, has a circular plan with eight arched sections that join together in the center of the circle. Also, since the restaurant's roof is a continuous concrete shell, not a grid of discrete members, the model's grid density should be high to better approximate a continuous shell. To create the representation of a continuous surface, the designer chooses a square grid that is significantly denser than the earlier example.

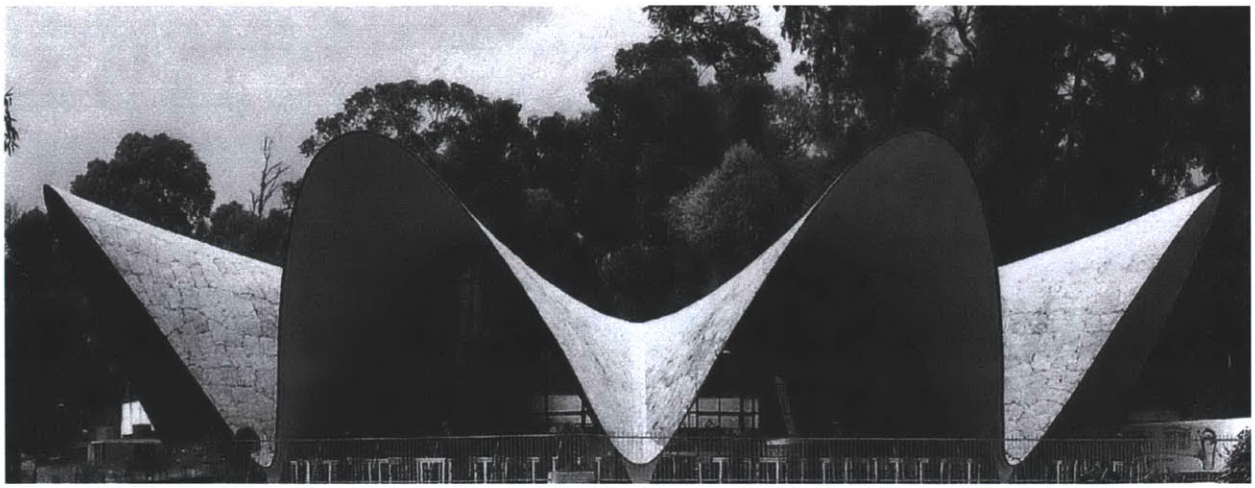


**Figure 5.8: Photo of Felix Candela's Los Manantiales Restaurant, Xochimilco, Mexico (Deutsches Museum n.d.)**

The designer begins by creating a circle and splitting it into eighths. The lines will represent the groins in the vault and will need to be stronger than the normal grid segments. Where the lines meet the circle is the fixed points where the shell touches the ground. Between the groins, the square grid is put in, with each section oriented 45 degrees from the next. The edge of the shell in plan will be straight, since the particle spring system can only work with straight- line segments, not curves. To keep from introducing irregularity into the square grid, no diagonal members can be put in; thus the grid cannot extend beyond the line between the fixed points without introducing bending. Finally the groin lines are split along the grid so that everything connects, and they are colored so they can be set as a different strength. The initial geometry is shown in Figure 5.9.

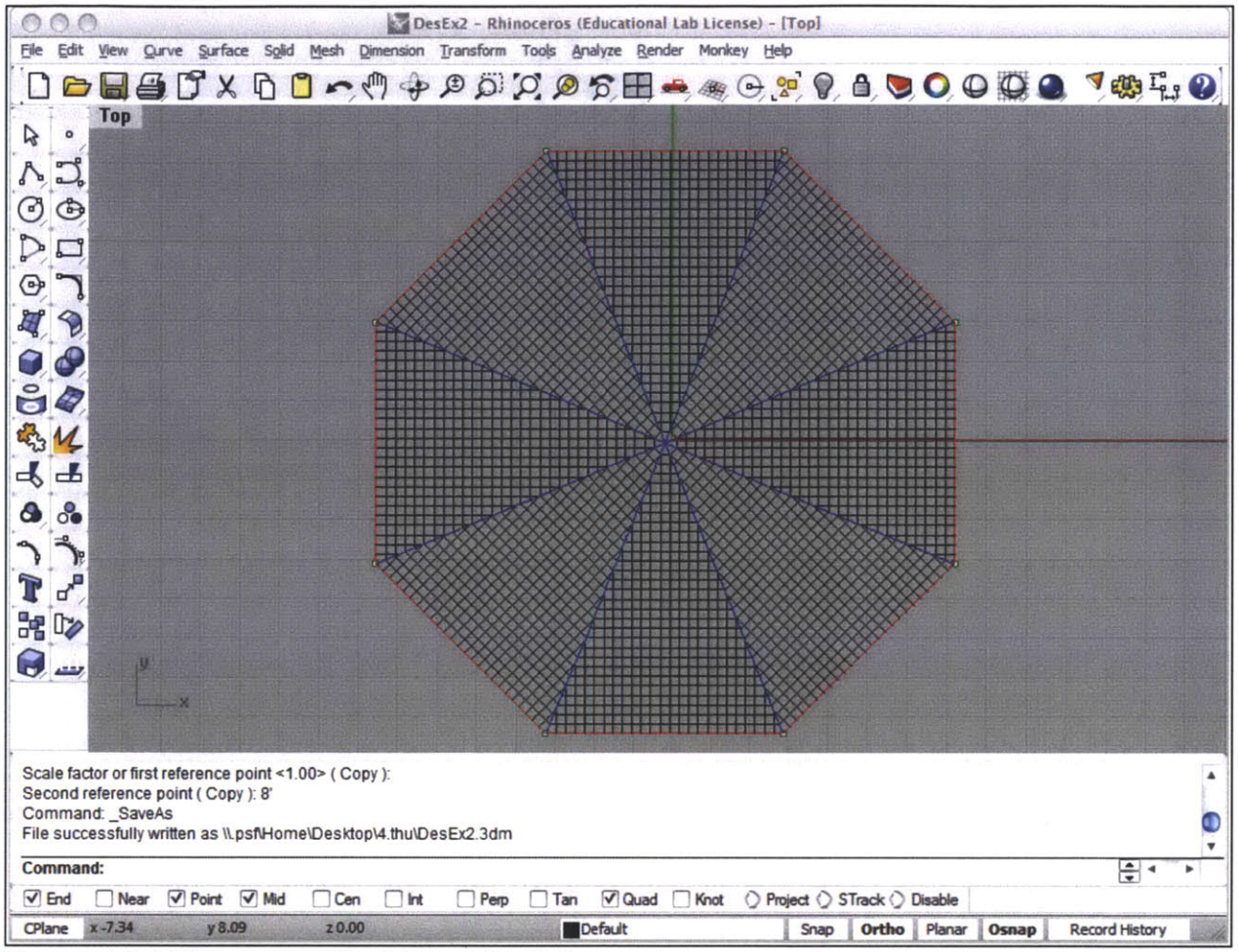



**Figure 5.9: Initial geometry of Los Manantiales shell**

After loading the initial geometry, the designer needs to change the relative spring strengths to achieve a shape similar to Candela's shell design, Figure 5.10. The blue groin lines need to be very stiff while the general grid needs to be relatively weak in order to have the desired curvature. The red edge lines should be the same strength as the general grid to give the best result.

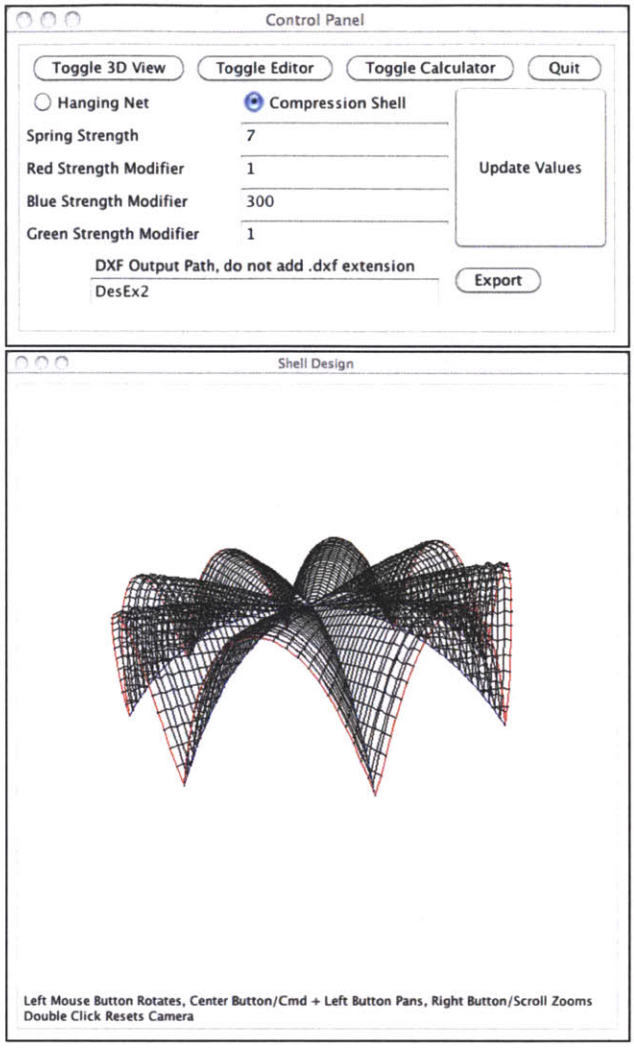



**Figure 5.10: Control Panel and Design View of ForceDesigner final shape for Los Manantiales Restaurant**

Because of the square grid, the system is no longer determinate. That means that multiple load paths are possible, and each spring is not necessarily in compression. This can be seen near the peaks of each arch section where the grid looks wrinkled. Where the peaks of Candela's hypar surface extend beyond the foundations in plan, that shape requires bending strength that the particle-spring model cannot introduce. Thus, the peaks do not extend out and the surface wrinkles instead. This highlights a drawback to this design method; it requires the designer to create an initial geometry that is either strictly determinate or that does not have areas that work

in bending. Overall, however, the designer succeeded in replicating the design and can now estimate the stresses in the surface by dividing the force in the members by the member spacing.

## 5.4 Adjoining Pavilion

For the final design example, we will go through the design process and use ForceDesigner to generate a number of form possibilities very quickly. A designer is confronted with a very open-ended design problem, to design a pavilion roof that is anchored from the roof of an existing building that reaches down to the ground. A design sketch is shown in Figure 5.11.

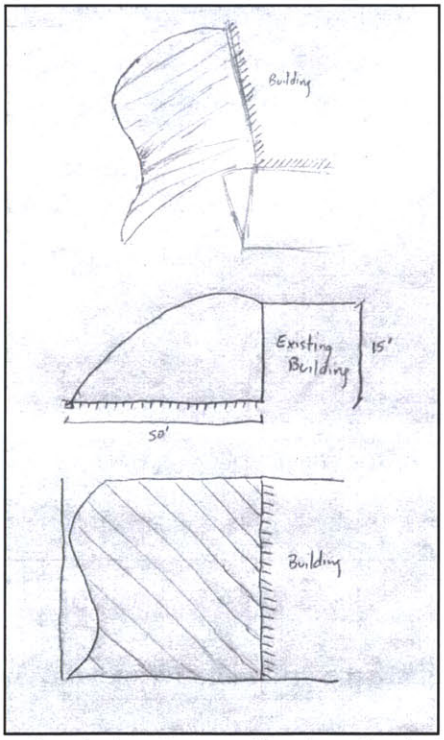
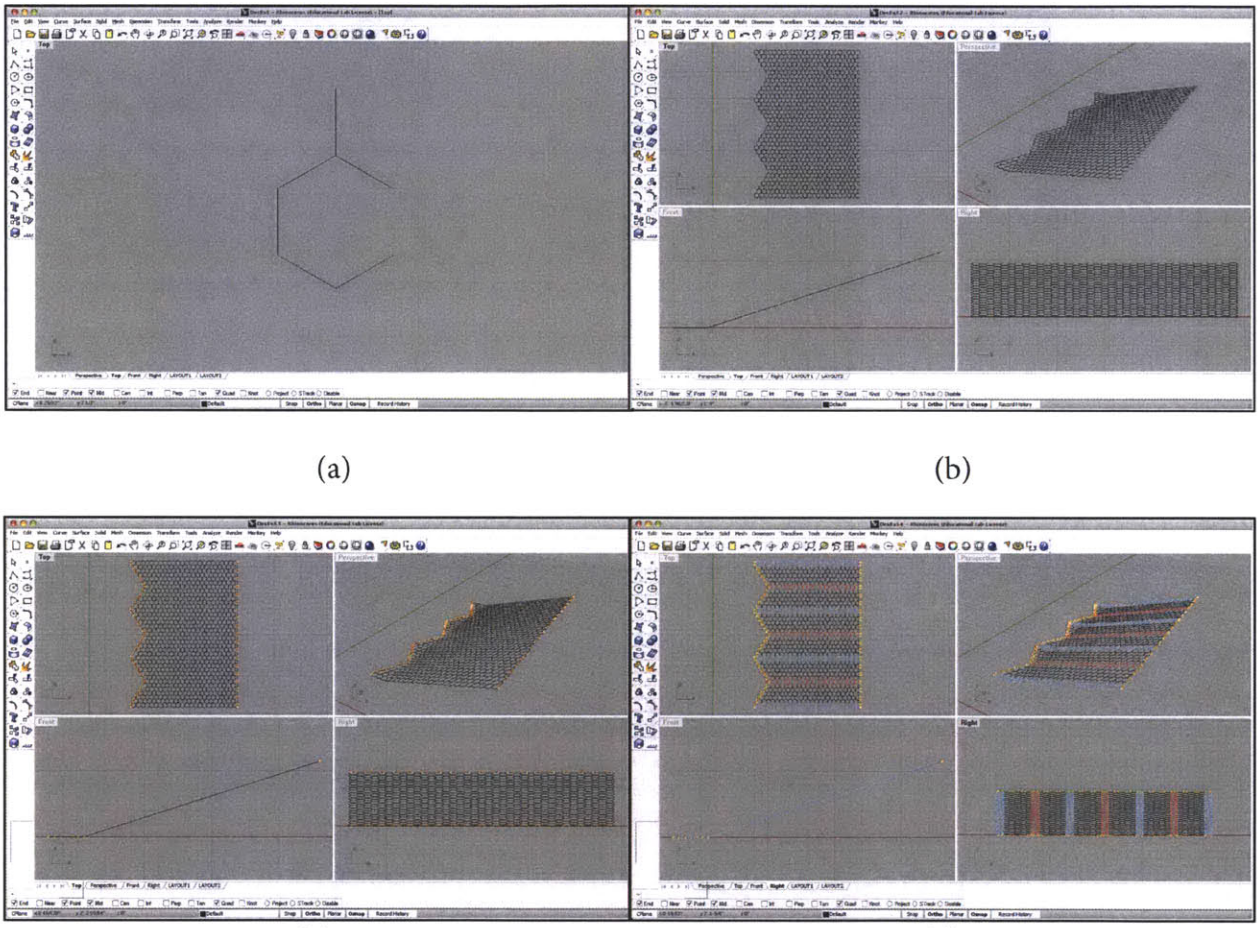



**Figure 5.11: Sketch showing specifications for design**

The design should include an undulating foundation on the ground for visual interest as well as lateral and buckling stability. An initial geometry can be generated very quickly using Rhinoceros using a hexagonal unit, as shown in Figure 5.12 a. The initial geometry has a rising elevation to represent the anchor on the roof of the existing building, and both sides are

completely fixed, as shown by the orange dots (Figure 5.12 b, c). The designer colors the model with the consideration that he may want to create ribs, or edge sections with variable strengths (Figure 5.12 d). Finally the design is exported and opened in *ForceDesigner*.

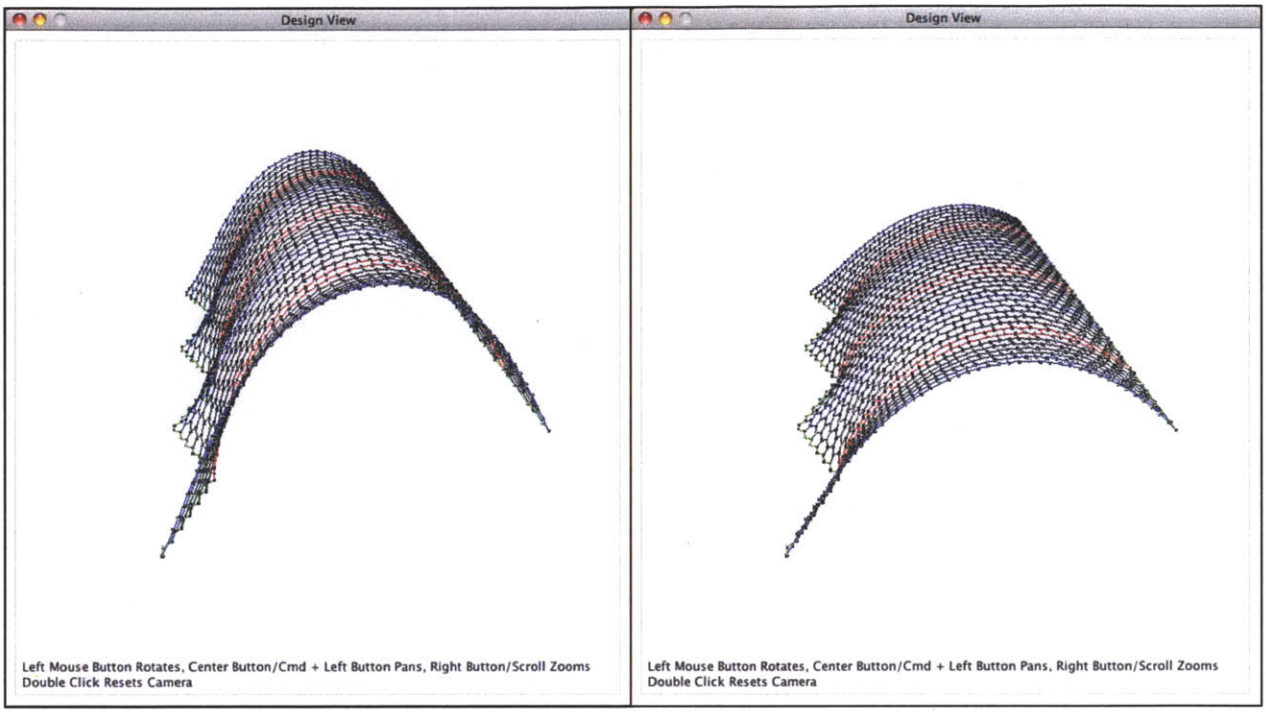


(a) (b)

(c) (d)

**Figure 5.12: Creating Initial geometry for pavilion. (a) Hexagonal base unit (b) unit arrayed and angled to fit geometry (c) Fixed points added (d) coloring for potential areas of different stiffness**

Upon opening the initial geometry, it must first be switched to a compression shell. The designer then sees that it looks much to tall, and so the spring strength must be increased from the base of 50. The designer increases the strength by a factor of 10, to 500, to decrease the height. The change is shown in Figure 5.13.




(a) (b)

**Figure 5.13: Views of Pavilion with Spring Strengths of (a) 50 and (b) 500**

The designer wants to emphasize the undulation of the rib though, and so increases the k value of the red and blue springs by another factor of 10. The result is shown in Figure 5.14.

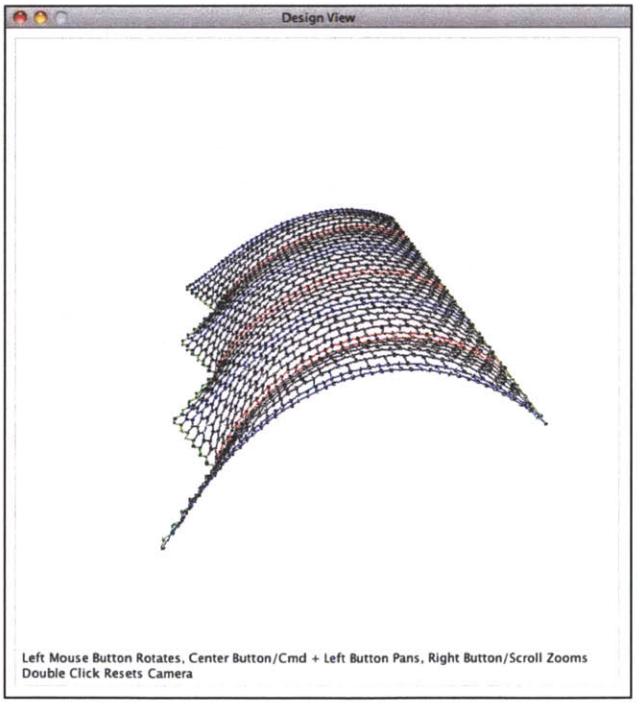
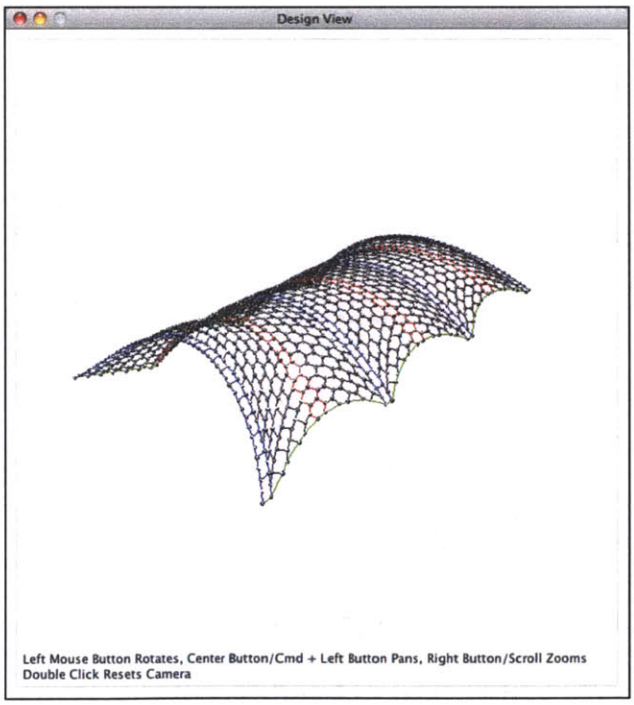
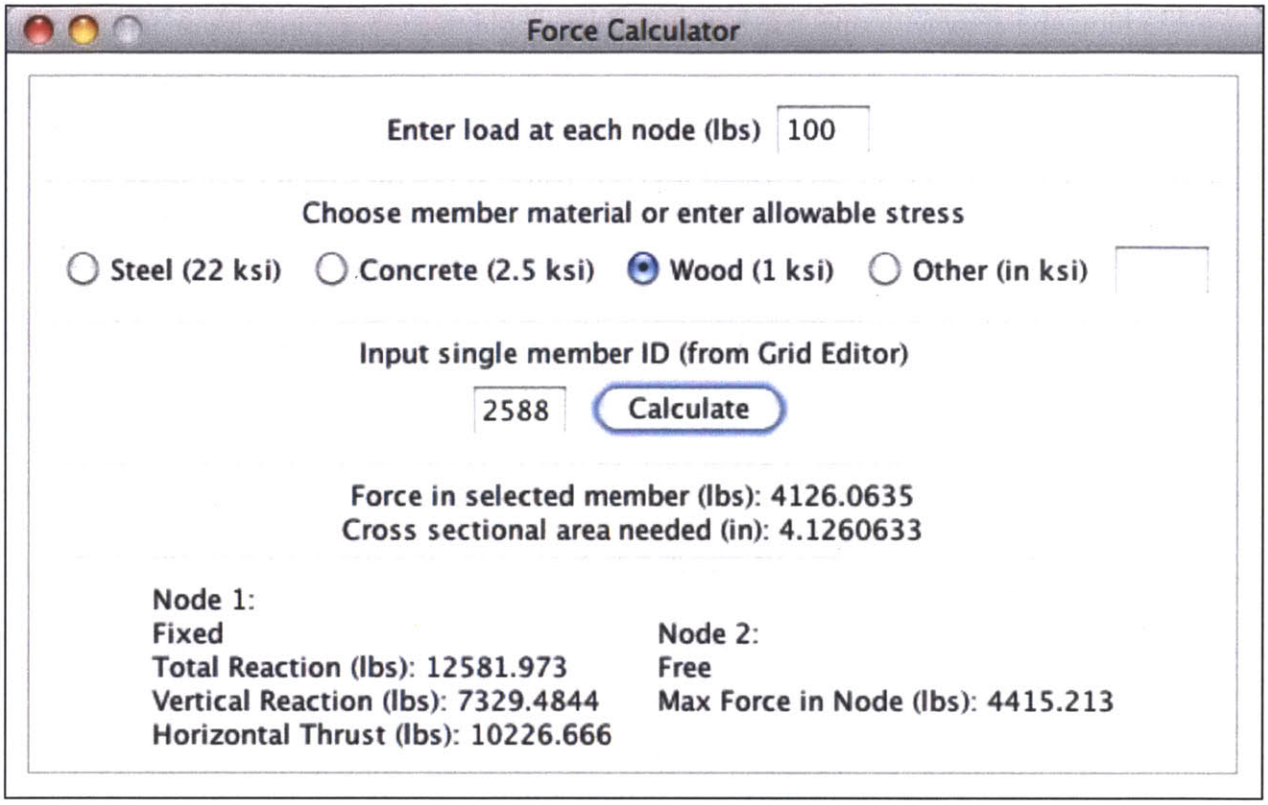



**Figure 5.14: View of Pavilion with increased rib stiffness**

Although happy with this design, the designer would like to introduce more light into the shell. This can be achieved by releasing some of the fixed nodes on the roof level, which would introduce side arches. The designer uses the editor panel to release a number of these nodes to create three large side arches (Figure 5.15 a). The engineer is concerned that this may put too much stress on the remaining ribs and put too much thrust at one point on the building, which it was not designed to take. The engineer plugs in one of the side arch members that abut the building, and the Force Calculator returns a horizontal thrust at the support node of over ten kips (Figure 5.15 b). This means at that point the existing wall is taking over 20 kips of thrust since the rib is composed of two members. He suggests that making the side arches smaller and having more of them will reduce the load to a more acceptable level.



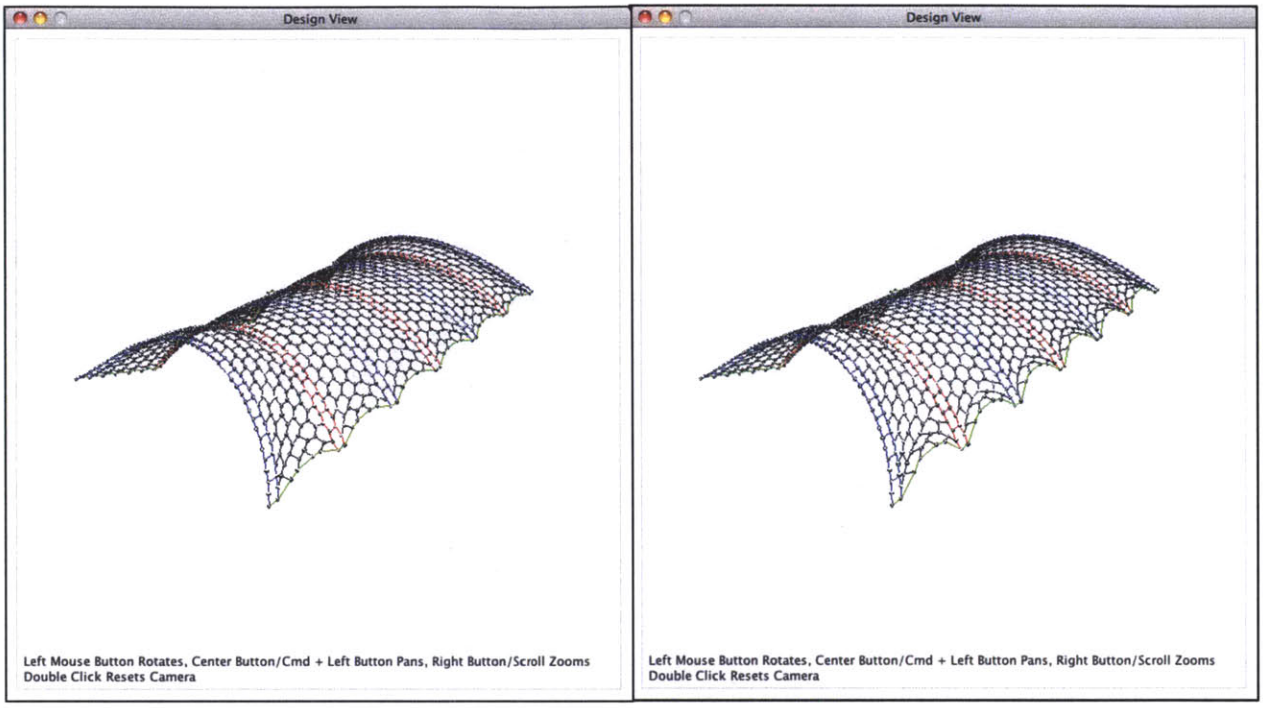

(a)




(b)

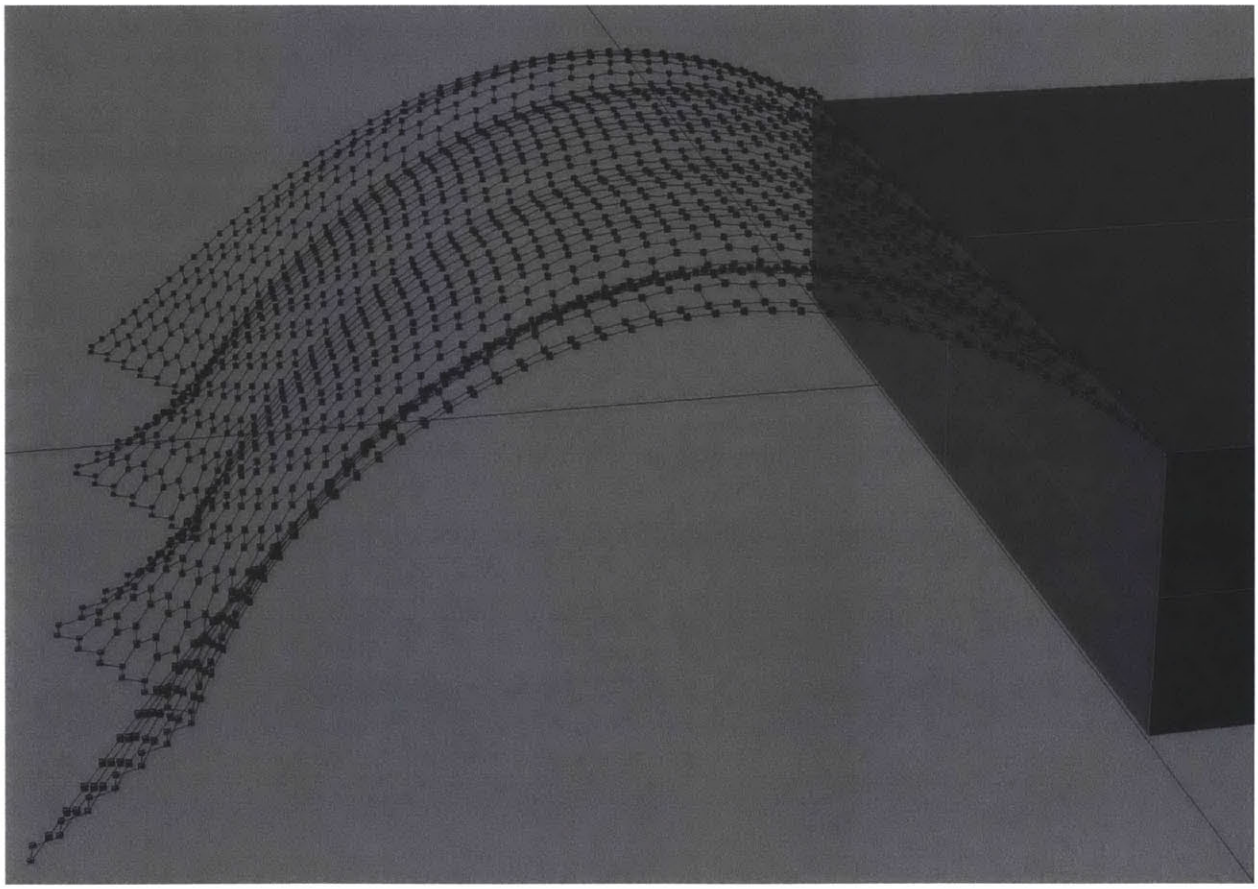
**Figure 5.15: (a) View of Pavilion with three side arches (b) Force calculator showing high horizontal thrust**

The designer uses the editor pane again to re-affix three sets of nodes. Upon following the same procedure, the thrust is much reduced, to 5.7 kips. The designer notices that these arches don't look as graceful as they could, and the engineer notes that raising the arch could reduce the load, allowing a more elegant structure. The designer changes the green spring modifier to 0.05 and the desired result is produced – the arches are taller and they gracefully protrude out from the rest of the shell. These results are shown in Figure 5.18. The result is that the arch member only needs to take 400 lbs. instead of 2.5 kips, meaning it can be much more slender. Also, the thrust on the building is reduced further, to less than five kips. The designer and engineer are pleased with this result and export the geometry, which is shown after being rendered in Rhinoceros in Figure 5.19. Overall this process took under half an hour to create an efficient and elegant shape for this pavilion, and it is in a form that can easily be further designed.

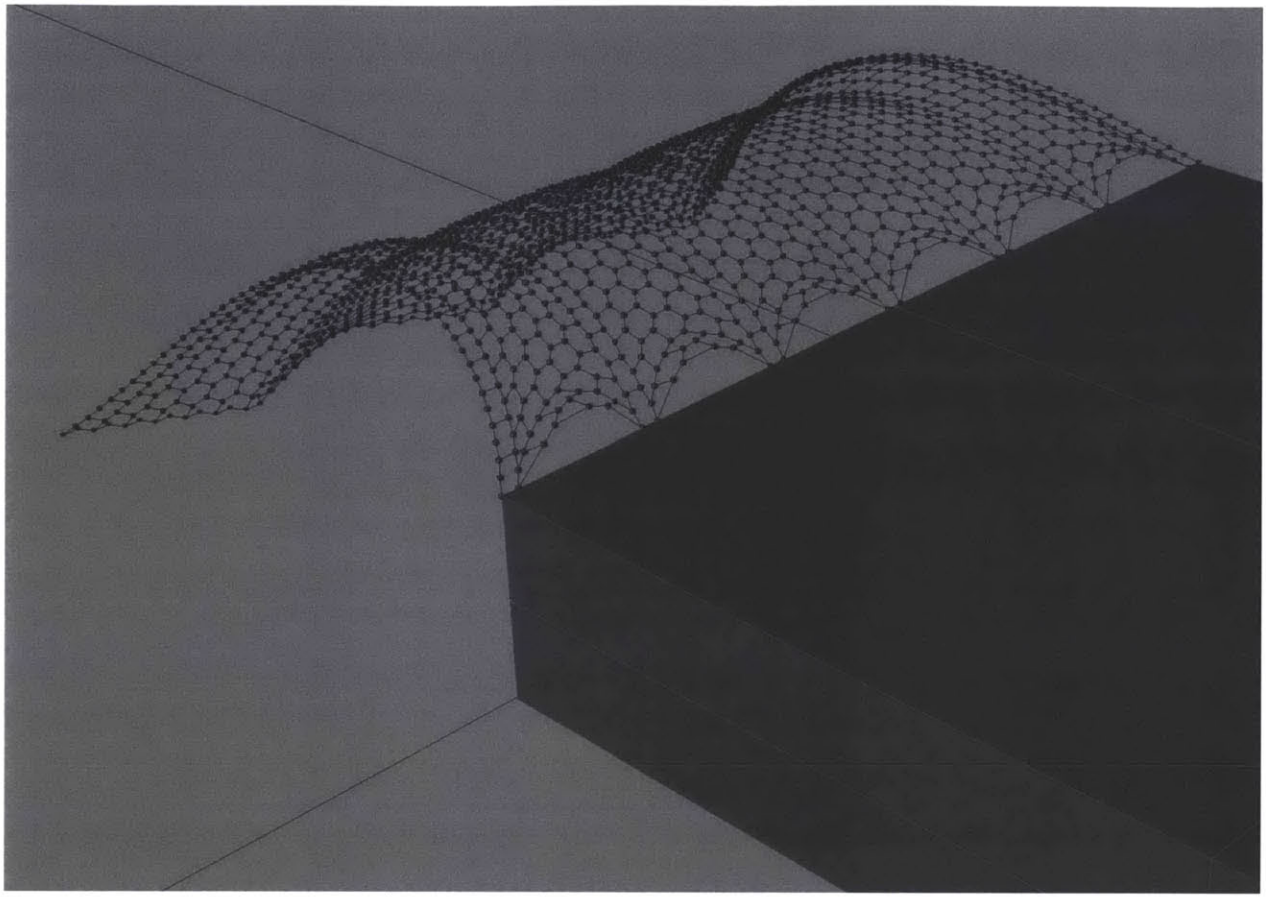



(a) (b)

**Figure 5.16: Views of Pavilion with six side arches, Green Strength Modifiers of (a) 1 and (b) 0.05**



(a)



(b)

**Figure 5.17: Final view of Pavilion in Rhinoceros with existing building (a) front view (b) side view**

## 5.5 Summary

This chapter has shown the versatility of *ForceDesigner* and the capability of the user to use its functions to quickly create varied designs. Each design was created in under thirty minutes, and can be exported for further design development. This chapter did not look at many of the more abstract options that are possible when the initial geometry is not uniform. The options for axial-force geometries are only limited by the imagination of the designer using the program.

# VI. Conclusion

## 5.1 Contributions

As seen in the previous chapter, *ForceDesigner* succeeds in implementing all of its initial goals. As a program, it contributes to the design community by allowing quick and easy early stage design for axial force structures. It improves functionality over previous design programs and adds new functionality and user interface.

Specifically, *ForceDesigner* allows the designer to create any sort of initial geometry. The user is not tied to a square grid or a single plane that points can be fixed to, any geometry in three dimensions with any number of nodes and springs can be used as an initial geometry for *ForceDesigner*. The program has new functionality in the different colored springs that allow four different spring k-values in a single model. This allows a much larger range of design options for the user without needing to create much more complex initial geometries to achieve the same thing. Within *ForceDesigner*, the designer has a limited ability to change the initial geometry by fixing or freeing any of the nodes, and changing the color of any of the springs. This means that the user will not need to continuously change the initial geometry to achieve different results, which is novel for this type of design program.

One of the largest contributions made by *ForceDesigner* is the engineering output tied into the design program. This program is not just for architects to design forms; it allows engineers to become much more involved as well. Having an early stage engineering output means that the engineers do not need to perform an FEM analysis for every design iteration. Too many times the engineer must try to salvage an inefficient design because the form had already been decided and the architect had already put in so much work without designing for the forces.

However, with *ForceDesigner*, the design results form the program can be made structurally efficient from the beginning, when the engineer should become involved. This is a major step forward in advancing the notion that architects and engineers need to work together to create beautiful and efficient structures.

## 5.2 Future Improvements

Although these contributions are significant, there are a few areas of *ForceDesigner* that could be improved. These are generally separated into improvements to the physical system behind the program, and improvements to the user interface.

One of the drawbacks of the implementation of *ForceDesigner* is that it is computationally expensive. Although in theory the program can design meshes of any scope, most computers quickly lose processing power when meshes become large. This is likely due to the inefficient way that the program updates the positions and forces in each spring. This is an area that could be fixed with help from a computer scientist. A more efficient algorithm for solving the mesh would allow much finer meshes, which could better approach the behavior of continuous membranes.

A second drawback in the model is its inability to model infinitely stiff springs. There is no way to model a chain mesh, or add members into the model that represent stiff links. The particle spring system crashes when the spring strengths get very high, above 10,000. In order to create these stiff links, a way to determine the force in the member beyond multiplying the extension and the strength would be necessary, which could be implemented with some more work.

A function that would be beneficial is the ability to set all members to be the same length. This would be useful in grid shell applications where the designers want uniform construction

units and modular sections for cladding. This would mean each member would have a dynamic k value so that the extension could be set equal to the others. This would require substantial modification of the underlying particle spring code, but would significantly improve the options for designers.

In terms of user interface, the Spring ID system is somewhat clunky. One way to make it better would be to implement a mouse control system, whereby clicking on a member would select it so it could be edited or analyzed. This would require much more experience coding Java interfaces to implement such a system. It would also be better if members could be added or deleted, but that could really only work once mouse control is implemented.

As for the Force Calculator, it could use a few improvements as well. First of all, a small change would be to figure out a way for the program to round to useful units. This is difficult because of the variety of scales the program could be used for, so setting an arbitrary value to round to could render the panel entirely useless. It could also benefit from being able to analyze multiple load cases, either within the Java program or a better link to an external FEM analysis.

## 5.3 Final Thought

This thesis attempts to improve design programs for efficient structures in the effort to encourage more of these structures to be built. By reducing the amount of material used to build, we are not only reducing the impacts of construction on the environment but producing beautiful structures as well. My hope is that this program, *ForceDesigner,* will lead to structures thoughtfully conceived out of concern for both form and forces.

-Alexander D.W. Jordan. (May 18, 2011)

# VII. Bibliography


Block, Philippe. *Thrust Network Analysis.* PhD Thesis, Cambridge, MA: Masachusetts Institute of Technology, 2009.

Chilton, John. "Heinz Isler's Inifinite Spectrum: Form-Finding in Design." *Architectural Design,* July 8, 2010: 64-71.

Deutsches Museum. *Félix Candela - Künstler der Konstruktion.* http://www.deutsches-museum.de/presse/presse-2011/felix-candela/ (accessed May 2011).

Feinberg, Jonathan. *PeasyCam.* 2008. http://mrfeinberg.com/peasycam/ (accessed May 12, 2011).

Fry, Ben, and Casey Reas. *Processing.* 2004. www.processing.org (accessed May 12, 2011).

Greenwold, Simon. *Simon Greenwold Portfolio.* http://acg.media.mit.edu/people/simong/ (accessed May 12, 2011).

Kilian, Axel. "Linking Hanging Chain Models to Fabrication." *DesignExplorer.* 2004. http://designexplorer.net/newscreens/cadenarytool/KilianACADIA.pdf (accessed May 2011).

Kilian, Axel, and John Ochsendorf. "Particle-Spring Systems for Structural Form Finding." *Journal of the International Association for Shell and Spatial Structures* 46, no. 2 (2005): 77-85.

McNeel and Associates. *Rhinoceros.* 2010. www.rhino3d.com (accessed May 12, 2011).

Otto, Frei, and Bodo Rasch. *Finding Form.* Munich: Deutscher Werkbund Bayern, 1995.

Scheck, H.J. "The force density method for form finding and computation of general networks." *Computer Methods in Applied Mechanics and Engineering,* 1974: 11-134.

Schenk, Mark. "On the shape of Cables, Arches, Vaults and Thin Shells." *University of Cambridge, Department of Engineering.* Novermber 10, 2009. http://www2.eng.cam.ac.uk/~ms652/files/IAStruct_cables_and_arches.pdf (accessed May 12, 2011).